IFIP Networking 2018

May 14-16, 2018
Zürich, Switzerland

Proceedings

IFIP Networking 2018 is technically co-sponsored by as well as hosted by:

Technically Co-Sponsored by IEEE computer society

Universität Zürich[UZH]

IFIP Networking 2018 gratefully acknowledges the support from the following patrons:

CSG Communication Systems Group

Kanton Zürich

Stadt Zürich

IFIP Networking 2018 General Chair's Welcome Message

Dear Participants of the 17th IFIP Networking 2018!

Exactly eighteen years ago the IFIP Networking conference history took off with its first instance on May 14, 2000 in Paris, France, forming at that point for the first time a joint event on formerly separated and since then integrated conferences on Broadband Communications (1995-1999), Performance of Communication Networks (1981-1998), and High Performance Networking (1987-1998). While IFIP Networking took place in the past for 11 times in Europe, for 3 times in North America, and twice in Asia, now the 17th IFIP Networking instance starts in Zürich, Switzerland. Thus, it is my pleasure to welcome you in my position as the IFIP Networking 2018 General Chair – and on behalf of the entire IFIP Networking Organizing Committee and Local Arrangements Team – to the 17th IFIP Networking 2018!

Since you are already holding these proceedings in your hands, you have arrived by now safely to the beautiful location of Zürich, either by plane, train, or even car. Therefore, please find this program now at your hands as a guideline to your technical session planning for IFIP Networking 2018, the conference venue, information on the social event, conference services, and last but not least on lots of activities in and around Zürich. As you will have noted, the public transport toward the Swissôtel in Zürich-Oerlikon is most likely the very shortest one you may have ever seen for any conference you may participate in! The 8 minutes travel by train or S-Bahn (so called fast, short distance trains in Switzerland) from the train station of the Zürich Unique Airport (ZRH) to Oerlikon Bahnhof (station) and only an additional 60 m footpath from there to the hotel's entrance is not only a great coincidence, it is Zürich's dedication to offer a frequent, timely, reliable, and efficient public transport, running besides trains, S-Bahn, and tram lines also buses and boats on the same ticket!

And you will learn during IFIP Networking 2018 that this public transport is also working for IFIP Networking's social program, supported by IFIP Networking's provision of sponsored local transport ticket for 24 hours. Besides for its use for the social event, please check out on such ticket-wide travel options within Zürich's travel zone 110! Main details are available on the route map provided in your participants' welcome package. And for sure, do not hesitate to talk to the Zürich local team or the reception desk for more detailed inquiries.

During IFIP Networking 2018 you can enjoy as a registered participant (please wear always your badge for room access) the welcome and key note sessions, technical sessions, the poster and demo session, the panel, a modern hotel of up-to-date conference facilities, complimentary WLAN access, and a great setting for work, coffee breaks, and joint lunches.

Such an event would not be possible without you, the attendees and paper authors! Thus, thank you for being with us and presenting your work. But additionally, it is the great help and drive of the IFIP Networking 2018 Organizing Committee, who made sure that all of you do see this great program and nice location. Special thanks go here to the three TPC Co-chairs of IFIP Networking 2018: Claudio Casetti, Fernando Kuipers, and James P. G. Sterbenz.

Last but not least it is my pleasure to address many thanks to the sponsors of IFIP Networking 2018 especially the Communication Systems Group CSG of the Department of Informatics IfI, the University of Zürich UZH, the Canton of Zürich, and the City of Zürich, who help to make your experience of IFIP Networking 2018 a unique one! Also, IFIP Networking 2018 is pleased to acknowledge the support by IFIP TC6 and the technical co-sponsoring by the IEEE Computer Society. Thus again, the entire Zürich local arrangements team wishes you a very warm welcome, a highly successful conference, productive "in-between" meetings, new contacts and ideas, and some great days out in and around the city of Zürich!

Burkhard Stiller

IFIP Networking 2018 General Chair

IFIP Networking 2018 Program Chairs' Welcome Message

It is with great pleasure that we welcome all attendees of the 17th edition of IFIP Networking in Zürich, Switzerland. The main objective of IFIP Networking is to bring together members of the networking community from both academia and industry, to discuss recent advances in the broad and quickly evolving fields of computer and communication networks, to highlight key issues, identify trends, and develop a vision for Future Internet technology, operation, and use.

As in the past editions, IFIP Networking 2018 has attracted a large number of papers in four main areas: (1) Network Architectures, Applications and Services, (2) Network Modeling and Analysis, (3) Network Security and Privacy, and (4) Wireless Networking.

This year, we received 225 submissions from a total of 41 countries. At a time when all major conferences are vying to attract research works, the high number of this year's submissions confirms that IFIP Networking is a well-established, reputed venue for high quality papers.

The main technical program is the result of a very selective, rigorous two-step peer-review process by experts in the areas of the conference. With all papers receiving an average of 4.4 reviews (at least 3 reviews and some of them up to 6 reviews) plus a meta-review summarizing the online discussion, we were able to guarantee that the works that made it into the main program are of outstanding quality. Overall, we accepted 55 full papers, which corresponds to an acceptance ratio of around 24%. In addition, 8 papers were accepted as short presentation papers. Within a fully separate submission track from the main papers, 4 demonstrations and 3 posters had been accepted, too, to enable an interactive discussion of operational systems and work-in-progress.

The technical program is complemented by three keynotes, namely "Blockchain for Cyber Physical Systems", "Flexibility Matters: On the Design and Evaluation of Softwarized Networks", and "Artificial Intelligence in Network Operations and Management", and a panel on "Security and Privacy in the Internet of Things", as well as by a poster/demo session.

We would like to thank all the members of the Organizing Committee who worked hard to make this conference a real success. First and foremost, we express our gratitude to Burkhard Stiller, the General Chair of IFIP Networking 2018, for his unflagging support and constant input. Of course, a special thanks is due to all authors who selected IFIP Networking 2018 as a platform to present their latest research results. Last, but not least, we would also like to acknowledge all TPC members and reviewers for their time in providing their rigorous, high-quality 1009 reviews.

Hopefully, you will find this program a source of inspiration of new ideas for your research and work. We encourage you to use the three days of the conference to discuss and create new connections with colleagues from all over the world. Finally, besides attending intellectually inspiring talks and discussions, make sure to enjoy Zürich!

Claudio Casetti
Fernando Kuipers
James P. G. Sterbenz

IFIP Networking 2018 Technical Program Co-chairs

Keynote I: Blockchain for Cyber Physical Systems

Salil Kanhere, University of New South Wales, Sydney, Australia

In Cyber Physical Systems (CPS), computing elements coordinate and communicate with sensors, which monitor cyber and physical indicators, and actuators, which modify the cyber and physical environment where they are run. Current CPS ecosystems rely on centralized, brokered communication models, otherwise known as the client-server paradigm. All devices are identified, authenticated, and connected through cloud servers. The data collected by devices is stored in the cloud for further processing. While this model has connected generic computing devices for decades and will continue to support small-scale CPS networks as we see them today, it will not be able to respond to the growing needs of the large-scale CPS ecosystems of tomorrow with billions of connected devices. Cloud servers will remain a bottleneck and point of failure that can disrupt the entire network. This is especially important as critical services and infrastructure such as healthcare, electric grids, logistics, and transportation become dependent on CPS. The current stove-piped architecture has also created isolated data silos, where users have limited control over their data and how it is used. Users have to trust the cloud and application providers and have no choice but to rely on their promises of security and availability.

This keynote will explore how the Blockchain (BC) technology has the potential to overcome the aforementioned challenges. BC is an immutable timestamp ledger of blocks that is used for storing and sharing data in a distributed manner. The data stored might be payment history, e.g., Bitcoin, or a smart contract or even personal data. In recent years, BC has attracted tremendous attention from practitioners and academics in different disciplines (including law, finance, and computer science) due to its salient features, which include decentralization, immutability, auditability, security, and privacy. Thus, the talk will specifically consider three key aspects of CPSes: (i) Internet of Things, (ii) Intelligent Transportation, and (iii) Supply Chain and will explain relevant concepts, will review the state-of-the-art, will present representative solutions, and will discuss open challenges.

Prof. Dr. Salil Kanhere received his M.S. and Ph.D. degrees, both in Electrical Engineering, from Drexel University, Philadelphia, U.S.A. He is an Associate Professor in the School of Computer Science and Engineering at UNSW Sydney, Australia. He is also a conjoint researcher at Data61 CSIRO, Faculty Associate at Institute of Infocomm Research Singapore, and on the advisory board of two technology start-ups.

His research interests include Internet of Things, pervasive computing, blockchain, crowdsourcing, data analytics, privacy, and security. He has published over 180 peer-reviewed articles and delivered over 20 tutorials and keynote talks on these research topics. He has received 4 Best Paper Awards. His research has been featured on ABC News Australia, Forbes, Wired, ZDNET, MIT Technology Review, IEEE Spectrum and other media outlets. Salil serves on the Steering Committee of IEEE LCN and is the program co-chair for IEEE WoWMoM 2018 and ACM MSWiM 2018. He regularly features on the organizing committee of a number of IEEE and ACM international conferences. He is on the Editorial Board of Elsevier's Pervasive and Mobile Computing and Computer Communications and on the Executive Committee of the IEEE Computer Society's Technical Committee on Computer Communications (TCCC). Salil is a Senior Member of both the IEEE and the ACM. He is a recipient of the Alexander von Humboldt Research Fellowship.

Keynote II: Flexibility Matters: On the Design and Evaluation of Softwarized Networks

Wolfgang Kellerer, Technical University of Munich (TUM), Germany

In order to address network dynamics and highly varying requirements, flexibility has emerged as a key property for networks to cope with increasing dynamics and to be prepared for future demands. Softwarized networks including concepts such as Network Virtualization, Software Defined Networking and Network Function Virtualization promise flexibility. However, so far flexibility is mainly used as a qualitative advantage for a certain design choice where the meaning of flexibility is varying a lot in literature. To provide a better understanding of how to design flexible networks, we propose a definition for flexibility and present an approach for a quantitative measure of flexibility in softwarized networks. In our proposal, we refer to flexibility as the ability to support new requests, e.g., changes in the requirements or new traffic distributions, in a timely manner. We illustrate with use case studies for function placement and SDN resilience, how this measure can be used to evaluate and compare different network designs quantitatively. To address adaptation time in flexible networks, we further present approaches to speed up the execution of algorithms based on machine learning. Examples include virtual network embedding and function placement. With our proposed approach for the definition and evaluation of flexibility, we intend to stimulate the discussion towards a more quantitative analysis of softwarized networks and beyond.

Prof. Dr. Wolfgang Kellerer is a full professor with the Technical University of Munich (TUM), Germany, heading the Chair of Communication Networks at the Department of Electrical and Computer Engineering. Before, he was for over ten years with NTT DOCOMO's European Research Laboratories. His last position was head of the research department for wireless communication and mobile networking.

His current research focuses on flexible networking based on SDN/NFV and wireless M2M networking towards 5G. He received his Dr.-Ing. degree (Ph.D.) and his Dipl.-Ing. degree (Master) from TUM, in 1995 and 2002, respectively. His research resulted in over 200 publications and 35 granted patents. In 2015, he has been awarded with a Consolidator Grant from the European Commission for his project FlexNets: "Quantifying Flexibility in Communication Networks". He is a member of ACM, VDE ITG, and a Senior Member of IEEE.

Keynote III: Artificial Intelligence in Network Operations and Management

Jürgen Quittek, NEC Laboratories Europe, Germany

Complexity of communication networks and their management and operations is continuously growing. At the same time, the capabilities of Artificial Intelligence (AI) technologies, in particular of deep machine learning, are growing rapidly and offer a way to deal with the complexity.

This keynote gives a brief overview of the history of AI technologies and shows how recent advancements provide powerful means of analysis and prediction suited to address several of today's challenges in network operations and management. Several examples illustrate the variety of potential applications of AI to networking. The outlook will address upcoming technology trends, such as reinforcement learning, representation learning, and automated reasoning.

Dr. Jürgen Quittek is Managing Director of the NEC Laboratories Europe in Heidelberg, Germany. He received his degree in communications engineering from RWTH Aachen in 1989 and his Ph.D. from Hamburg University of Technology (TUHH) in 1996. After a postdoctoral year in Berkeley, California, he joined the NEC Laboratories in 1997. In 2000 he was a visiting professor at Freie Universität Berlin.

He conducted research in the areas of neural networks, network management, data security, software-defined networking, energy-efficient communications, and 5G mobile networks, and he served as TCP chair and member of many conferences and workshops. As working group chair, rapporteur, and author he contributed to communication standards at ETSI, IETF, and ONF. His current research interests also include artificial intelligence and the Internet of Things (IoT).

Panel: Security and Privacy in the Internet of Things

Panel Chair: James P. G. Sterbenz, The University of Kansas, U.S.A. and Lancaster University, U.K.

The Internet of Things (IoT) has become not only a hot topic for research, but as usual, is being deployed before we understand the implications of this technology, and without developed usability, security, privacy, resilience, survivability, controllability, accountability, and manageability.

This panel discusses whether IoT can or will be deployed with acceptable security and privacy for users and society, and whether lessons can be learned from the current ubiquitous mobile Internet. Are we racing towards a machine-assisted utopia or a machine-controlled dystopia?

IFIP Networking 2018 Organizing Committee

General Chair

Burkhard Stiller, University of Zürich, Switzerland

Technical Program Co-chairs

Claudio Casetti, Politecnico di Torino, Italy

Fernando Kuipers, Delft University of Technology, The Netherlands

James Sterbenz, The University of Kansas, U.S.A. and Lancaster University, U.K.

Publication Chairs

Christian Doerr, Technische Universiteit Delft, The Netherlands

Local Arrangements

Barbara Jost, University of Zürich, Switzerland

Web and Publicity Chair

Corinna Schmitt, University of Zürich, Switzerland

Final Proceedings Preparation for IFIP Digital Library

Muriel Franco, University of Zürich, Switzerland

Local Team

Bruno Rodrigues, University of Zürich, Switzerland

Eder Scheid, University of Zürich, Switzerland

Sina Rafati, University of Zürich, Switzerland

Dominik Bünzli, University of Zürich, Switzerland

Erica Maurer, University of Zürich, Switzerland

Steering Committee

Jordi Domingo-Pascual, Universitat Politècnica de Catalunya (UPC), Spain (Chair)

Andrea Passarella, IIT-CNR Pisa, Italy

Aiko Pras, University of Twente, The Netherlands

Henning Schulzrinne, Columbia University, USA

Jozef Wozniak, Gdansk University of Technology, Poland

IFIP Networking 2018 Technical Program Committee

Nadjib Aitsaadi, LIGM/CNRS, France
Özgü Alay Erduran, Simula Research Lab, Norway
Kevin Almeroth, University of California, Santa Barbara, U.S.A.
Nils Aschenbruck, University of Osnabrück, Germany
Stefano Avallone, University of Naples, Italy
Vaibhav Bajpai, Technische Universität München, Germany
Marinho Barcellos, Federal University of Rio Grande do Sul, Brazil
Suzan Bayhan, Technische Universitat Berlin, Germany
Nicole Beckage, University of Kansas, U.S.A.
Robert Bestak, Czech Technical University, Prague, Czech Republic
Christian Bettstetter, University of Klagenfurt, Germany
Andrea Bianco, Politecnico di Torino, Italy
Gergely Biczók, Budapest University of Technology and Economics, Hungary
Fernando Boavida, University of Coimbra, Portugal
Alessio Botta, University of Napoli Federico II, Italy
Raouf Boutaba, University of Waterloo, Canada
Raffaele Bruno, IIT-CNR, Italy
Anna Brunstrom, Karlstad University, Sweden
Milind Buddhikot, Bell Labs, U.S.A.
Claudia Campolo, University Mediterranea of Reggio Calabria, Italy
Antonio Capone, Politecnico di Milano, Italy
Augusto Casaca, INESC-ID, Portugal
Ignacio Castro, Queen Mary University of London, U.K.
Matteo Cesana, Politecnico di Milano, Italy
Egemen Cetinkaya, Missouri University of Science and Technology, U.S.A.
Marco Conti, IIT-CNR, Italy
Italo Cunha, Universidade Federal de Minas Gerais, Brazil
Stefan Dietzel, Humboldt-Universität zu Berlin, Germany
Jordi Domingo-Pascual, UPC, Spain
Benoit Donnet, Université de Liège, Belgium
Idilio Drago, Politecnico di Torino, Italy
Lars Eggert, NetApp, Germany
Joachim Fabini, Vienna University of Technology, Austria
Marwan Fayed, University of Stirling, U.K.
Laura Marie Feeney, Uppsala University, Sweden
Simone Ferlin-Oliveira, IBM Oslo, Norway
Markus Fidler, Leibniz Universität Hannover, Germany
Marco Fiore, IIT-CNR, Italy
Victoria Fodor, KTH Royal Institute of Technology, Sweden
Klaus-Tycho Foerster, Aalborg University, Denmark
Bela Genge, Petru Maior University of Tirgu Mures, Romania
James Gross, KTH Royal Institute of Technology, Sweden
Paola Grosso, University of Amsterdam, The Netherlands
Deke Guo, National University of Defence Technology, China
Al Harris, University of Illinois at Urbana-Champaign, U.S.A.
David Hausheer, OVGU Magdeburg, Germany
Boudewijn Haverkort, University of Twente, The Netherlands
Poul Heegaard, Norwegian University of Science and Technology, Norway
Markus Hofmann, Bell Labs/Alcatel-Lucent, France
Karin Hummel, JKU Linz, Austria
Adele Lu Jia, Delft University of Technology, The Netherlands

Hongbo Jiang, Huazhong University of Science and Technology, China
Lei Jiao, University of Oregon, U.S.A.
Gunnar Karlsson, KTH Royal Institute of Technology, Sweden
Hana Khamfroush, University of Kentucky, U.S.A.
David Koll, University of Goettingen, Germany
Kimon Konto Vasilis, NCSR Demokritos, Greece
Wi Koong Chai, Bournemouth University, U.K.
Yevgeni Koucheryavy, Tampere University of Technology, Finland
Dimitrios Koutsonikolas, University at Buffalo, SUNY, U.S.A.
Udo Krieger, Otto-Friedrich-University Bamberg, Germany
Franck Le, IBM T. J. Watson, U.S.A.
Guy Leduc, University of Liege, Belgium
Fengjun Li, The University of Kansas, U.S.A.
Jörg Liebeherr, University of Toronto, Canada
Xuan Liu, AT&T Labs Research, U.S.A.
Wenping Liu, Huazhong University of Science and Technology, China
Alex Liu, Michigan State University, U.S.A.
Samantha Lo, Google Inc., U.S.A.
Renato Lo Cigno, University of Trento, Italy
Leonardo Maccari, University of Trento, Italy
Olaf Maennel, Tallinn University of Technology, Estonia
Francesco Malandrino, Politecnico di Torino, Italy
Zoubir Mammeri, Paul Sabatier University, France
Vincenzo Mancuso, IMDEA Networks Institute, Spain
Victoria Manfredi, Wesleyan University, U.S.A.
Angelos Marnerides, Lancaster University, U.K.
Carmen Mas, Technical University of Munich, Germany
Daniel Menasché, Federal University of Rio de Janeiro, Brazil
Enrico Natalizio, Université de Technologie de Compiégne, France
Ilkka Norros, VTT Technical Research Centre of Finland, Finland
Jun Ogawa, Fujitsu Lab, U.S.A.
Sharief Oteafy, DePaul University, Canada
Jörg Ott, Technische Universität München, Germany
Philippe Owezarski, LAAS-CNRS, France
Elena Pagani, University of Milano, Italy
Marc-Oliver Pahl, Technische Universität München, Germany
Christos Papadopoulos, Colorado State University, U.S.A.
Andrea Passarella, IIT-CNR, Italy
Veljko Pejovic, University of Ljubljana, Slowenia
Colin Perkins, University of Glasgow, U.K.
Harry Perros, North Carolina State University, U.S.A.
Andreas Peter, University of Twente, The Netherlands
Jonathan Petit, OnBoard Security, U.S.A.
Dimitris Pezaros, University of Glasgow, U.K.
Mario Pickavet, Ghent University - iMinds, Belgium
Ana Pont, Universitat Politécnica de València, Spain
Aiko Pras, University of Twente, The Netherlands
Bruno Quoitin, University of Mons, Belgium
Vijay Rao, Delft University of Technology, The Netherlands
Peter Reichl, University of Vienna, Austria
Gábor Rétvári, Budapest University of Technology and Economics, Hungary
Björn Richerzhagen, Technische Universität Darmstadt, Germany
Claudio Rossi, ISMB, Torini, Italy

Pablo Gabriel Romero, Universidad de la República, Uruguay
Ramin Sadre, Université catholique de Louvain, France
Dola Saha, University at Albany, SUNY, U.S.A.
José Jair Santanna, University of Twente, The Netherlands
Stefan Schmid, Aalborg University, Denmark
Jens Schmitt, University of Kaiserslautern, Germany
Jürgen Schönwälder, Jacobs University Bremen, Germany
Henning Schulzrinne, Columbia University, U.S.A.
Cheta Singhal, IIT Kharagpur, India
Christoph Sommer, Paderborn University, Germany
Yang Song, IBM Research, U.S.A.
Vasilis Sourlas, University College London, U.K.
Otto Spaniol, RWTH Aachen University, Germany
Razvan Stanica, INSA Lyon, France
Ioannis Stavrakakis, National and Kapodistrian University of Athens, Greece
Moritz Steiner, Akamai, U.S.A.
Guang Tan, SIAT, Chinese Academy of Sciences, China
Y.C. Tay, National University of Singapore, Singapore
Chen Tian, Nanjing University, China
Gareth Tyson, Queen Mary, University of London, U.K.
Steve Uhlig, Queen Mary, University of London, U.K.
Dan Wang, Wichita State University, U.S.A.
Qing Wang, KU Leuven, Belgium
Wei Wang, Nanjing University, China
Klaus Wehrle, RWTH Aachen University, Germany
Michael Welzl, University of Oslo, Norway
Cedric Westphal, Huawei Innovation Center, U.S.A.
Joerg Widmer, IMDEA Networks Institute, Spain
Sabine Wittevrongel, Ghent University, Belgium
Lars Wolf, Technische Universität Braunschweig, Germany
Tilman Wolf, University of Massachusetts, U.S.A.
Fan Wu, Shanghai Jiao Tong University, China
Di Wu, Sun Yat-sen University, China
Kui Wu, University of Victoria, Canada
Fu Xiao, Nanjing University of Posts and Telecommunications, China
Zhi-Li Zhang, University of Minnesota, U.S.A.
Rong Zheng, McMaster University, Canada
Michael Zink, University of Massachusetts Amherst, U.S.A.

Table of Content

Rasch analysis of HTTPS reachability

George Michaelson
APNIC*
ggm@apnic.net

Matthew Roughan Jonathan Tuke
UoA‡ and ACEMS†
{matthew.roughan,simon.tuke}@adelaide.edu.au

Matt P. Wand
UTS§ and ACEMS†
matt.wand@uts.edu.au

Randy Bush
IIJ¶
randy@psg.com

***Abstract*—The use of HTTPS as the only means to connect to web servers is increasing. It is being pushed from both sides: from the bottom up by client distributions and plugins, and from the top down by organisations such as Google. However, there are potential technical hurdles that might lock some clients out of the modern web. This paper seeks to measure and precisely quantify those hurdles in the wild. More than three million measurements provide statistically significant evidence of degradation. We show this through statistical techniques, in particular Rasch analysis, which also shows that various factors influence the problem ranging from the client's browser, to their locale.**

I. Introduction

There is a growing push for "HTTPS Everywhere," where HTTPS, or more exactly HTTP over TLS (Transport Layer Security), is a more secure form of the standard Hyper-Text Transfer Protocol. It is more secure in that it provides:

1. server authentication using certificates, *i.e.,* a server can prove its identity;
2. a private communications channel, *i.e.,* it prevents eaves-dropping; and
3. data integrity, *i.e.,* it prevents standard man-in-the-middle attacks.

HTTPS Everywhere is the ubiquitous use of HTTPS in preference to HTTP for all services, not only those specifically requiring a secure connection.

The Electronic Frontier Foundation (EFF) is promulgating a browser extension to this effect [1] as a defence against spying, *e.g.,* from nation states in the post-Snowden era. Google supports the idea [2], and has announced that they will give search-rank priority to HTTPS sites [3]. And the increase in the number of clients accessing the Internet through wireless connections mandates encryption at the connection level. Reactions include the HTTPS-Only Standard [4], for the US Federal Government.

There is a performance cost documented [5]–[7] as far back as the 1990s. This cost arises primarily because the certificate exchange requires an additional round trip at the start of a connection. However, most HTTP requests don't require a full handshake, and with modern hardware the cryptography overhead is not critical. For example Doug Beaver from Facebook, stated *"We have found that modern software-based TLS implementations running on commodity CPUs are fast enough to handle heavy HTTPS traffic load without needing to resort to dedicated cryptographic hardware. We serve all of our [Facebook's] HTTPS traffic using software running on commodity hardware."* [8].

So on the face of it, HTTPS Everywhere is a "no brainer." There is even an "HTTP Shaming" web page.

HTTPS Everywhere seems to be happening. StatOperator [9] reported that the number of (the top million) sites using HTTPS as the default increased from around 103 to 219 thousand from 2016 to 2017. Google reports client usage statistics [10], [11], and they show similar steady growth from 2015 to the present.

However, there is an important question to answer before we convert the entire Internet to HTTPS: *Will there be people who are stranded behind port 80?*

We know that HTTPS is not an issue for many people (the current large-scale deployments of HTTPS prove that it mostly works), but there could be locations, or users of specific equipment that face challenges. Detailed reasons are given in Section II. They range from concern about the quality of the technology, to the rejection of compromised connections.

In this paper we provide evidence to inform the technical and policy debate concerning the deployment of secure web services, by measuring whether users can access HTTPS in the wild. We collected 3.3 million observations using APNIC's web advertising infrastructure [12], from which we found that there is sufficient evidence to show that HTTPS is *not* easily accessible to all Internet users.

A secondary concern of this paper is the statistical rigour necessary to allow such a statement to be made with confidence. The proportion of users that failed to make an HTTPS connection in our study was small. It has been common in the past to simply report numbers, but our goal is to provide statistically confident statements, despite a noisy and faint signal. The ability to detect such faint signals is important — a mere 0.1% of users now represents millions of individuals. We do so using both standard statistical tests, and a tool that has not been previously used in Internet measurement, but which may find many other applications: Rasch analysis [13], [14].

We found statistically significant evidence that there are clients that find HTTPS connections harder to complete than HTTP, and that this difficulty was influenced by origin autonomous system, browser, country of origin, and operating system, suggesting a range of causes.

*APNIC, South Brisbane, 4101, QLD, Australia.
†ARC Centre of Excellence for Math. & Stat. Frontiers.
‡University of Adelaide, Adelaide, 5005, SA, Australia
§University of Technology Sydney, Ultimo, 2007, NSW, Australia.
¶Internet Initiative Japan (IIJ) Research Lab, Tokyo, Japan.

ISBN 978-3-903176-08-9

II. Background and Related Work

A. Experimental Context

Simple web services with no protection against snooping or identity are typically conducted over TCP port 80, using the HTTP protocol. We call this 'port 80' service or HTTP.

Web services which are protected by Transport Layer Security (TLS) are usually conducted over TCP port 443, commonly called 'port 443' or HTTPS.

There have been many studies of HTTPS. However, they have focused on two main topics.

1. The certificate landscape, *e.g.,* see [15]–[17], in which the problems with certificate distribution have led to security holes, and consequent fixes[1].
2. Comparisons between HTTP and HTTPS performance, looking primarily at their latency difference, *e.g.,* see [5], [6], but also considering communications overhead and energy consumption [7].

As a consequence of using HTTPS, an additional handshake is needed to establish a connection. There can be no effective proxy-caching of the content, and filtering (*e.g.,* by firewalls) is hampered. HTTPS also uses cryptography which induces extra computational (and hence energy) costs, which may be trivial on a modern computer, but may be important on battery-operated devices, such as mobile phones.

A deeper consequence of the additional layer of complexity is the potential for failures. Surprisingly, studies of HTTPS appear to assume basic reachability, or more correctly, they appear to assume that HTTPS reachability, while perhaps not perfect, will be no worse than HTTP. However, it is not obvious that this will be so. A prominent browser maker asked if the Asia-Pacific Network Information Centre (APNIC) Labs ad-based measurement system [12] could see if a statistically significant number of users were unable to access TLS protected web resources.

So, what are the possible concerns? They range widely; the following is an incomplete list.

1. A browser or OS may be too old to perform TLS at the current specification. The web server used in this experiment did not offer older approaches, such as RC4 cryptography, so there is a chance that pre-TLS 1.x browsers will fail. However, the older standards are no longer considered secure, and it is our view that providing a false appearance of security is worse than providing none. It might be tempting to tell users to "catch up", but this is infeasible on mobile networks that sell captive locked phones left behind on "old cold" protocol variants.
2. Some modern browsers use intermediate systems to speed up or cache data. Opera, for instance, deployed a world-wide "anycast" cloud of intermediates to offer speed-up services, performing tasks such as JPG compression, to make the web faster. It is possible that this service notionally works with TLS, but that it works badly for flows it has in port 80 that move up into TLS because the state doesn't exist. Other well-meaning intermediary systems might break such up-lifts.
3. The additional overhead of the extra handshake makes the session more vulnerable to network problems, and hence less stable.
4. TLS protects against the threat of bad actor man-in-the-middle attacks. If an on-path attacker intercepts the session and attempts to hijack an aspect of the content, TLS should prevent the flawed connection. However, if such attacks are prevalent, they become DoS attacks on the HTTPS service.
5. A firewall along the path might block encrypted traffic as a matter of course. Though most firewalls allow port 80 traffic, they sometimes block all other ports. This might be considered misconfiguration, but misconfiguration is not uncommon [18].
6. Firewalls or other middle-boxes may perform their own hijacking of a connection through installed certificates on user machines.
7. Flaws in implementations or configuration [19], [20].

Our approach uses a cross-site reference within an advertisement in order to create a measurement. The underlying idea is not new. It has been used to measure DNSSEC and IPv6 deployment, among other features, *e.g.,* [21] (or for a more general review see [22]). However, our approach differs in several respects from [22]. The most important is that it performs a pair of measurements: a control based on HTTP, and an actual measurement of HTTPS, the focus. As far as we are aware, past studies have typically lacked a control, and therefore have been hard to interpret statistically.

However, APNIC's measurement infrastructure also differs from other approaches in that we use (paid) web-advertising to instantiate the tests (details below). Additionally, all fetches are to an APNIC-managed server, avoiding the major ethical controversies of past experiments (see Section III-D for more discussion of this issue).

B. Simple Statistical Background

Here we lay out the key statistical background. The material is somewhat tutorial, but as these techniques are not commonly applied in the Internet measurement context, we feel it is valuable to be precise about the methods and their rationales. We start by defining terminology:

Observation: the collected responses of a single client's connection attempts (see Section III-A for details).

Sample: the set of all observations.

Measurement: a particular feature of an observation, for instance, whether a successful HTTPS GET was completed. We also call these *response variables*, and denote them by random variables (RV) $Y^{(j)}$, where $j \in \{\text{HTTP}, \text{HTTPS}\}$ is the *treatment*, and the measurement

$$Y^{(j)} = \begin{cases} 1, & \text{if measurement } j \text{ succeeds,} \\ 0, & \text{otherwise.} \end{cases}$$

The sample is the collection of instances $\{y_i^{(j)}\}$ of this RV.

[1] TLS security is predicated on valid certificates, and there have been significant problems resulting from this weakness in the past. However, Certificate Transparency mitigates many of these issues [16].

Test: a statistical test applied to the data.
Categorical variable: one that takes a set of discrete values.
Predictor: a variable, also called a *covariate*, whose value *may* influence the outcome of the measurements.

We make a distinction here between a measurement and a test, the latter meaning a *hypothesis test* to discriminate between a *null-hypothesis* H_0 and its alternative H_1. The advantage of a hypothesis test is that it is consistent and repeatable with strict, precisely-defined assumptions and interpretation. Through their use we can avoid making common errors, such as over-interpreting limited evidence.

The test is conducted with respect to a significance level, α, chosen at the outset of the experiment. Here we use the common choice of $\alpha = 0.05$. This sets the Type I error probability (the chance we reject H_0 incorrectly). The Type II error probability (the probability we fail to reject H_0 when it is false) is determined by the power of the test on the particular data. Thus we cannot control for it, but can ensure it is small by providing enough observations.

We calculate a test statistic, determine from this a p-value, and then reject the null-hypothesis if the p-value falls below α. The common interpretation of the p-value is that it is the probability, given the null-hypothesis is true, of observing the at least the given test statistic. Hence, a small p-value can be taken as evidence that the null-hypothesis is invalid. However, we must be careful of this interpretation, because of the underlying statistical nature of the problem.

When the null-hypothesis is true, we would expect to see a uniform distribution of p-values over a set of repeated experiments, which includes some values $<\alpha$, leading to Type I errors. In order to avoid incorrect inferences in repeated experiments, we should try to control the *Family-wise error rate* (FWER) not the *Per-comparison error rates* (PCER). We shall do so here using the *Bonferroni correction* [23], in which α is divided by the number of tests in the family. We should note that this is rather conservative, and that there are other more complex procedures available [23], but we deliberately use a conservative FWER here.

Our experiment is a *matched pairs* experiment. That is, the pair of measurements is conducted on the same client, the question of interest being whether some users have more trouble with HTTPS than HTTP. This cannot be answered simply by comparing the proportions of successes for each measurement, because in a matched pair experiments the measurements are very likely correlated. Simply plotting the two probabilities, while useful in an explanatory sense, would not take these correlations into account.

However, the inclusion of our control experiment makes it possible to ask this question in the formal context of hypothesis testing using *McNemar's test* [24], with hypotheses:

- H_0 is that $p_1 = p_2$; and
- H_1 is that $p_1 \neq p_2$;

where p_j is the probability that the jth measurement of any particular observation is successfully completed. Rejecting the null implies significant evidence that the difficulty of the two measurements is different.

C. Rasch Modelling and Analysis

Hypothesis tests are an important starting point, but they only tell us "if" but not "how much?" This paper further proposes the use of Rasch analysis, an approach within the broader area of *Item Response Theory* (IRT). It is best illustrated by its application to the analysis of exams. An exam consists of a list of m questions, performed by n students. Each student answers each question either correctly, or not, forming the binary response variables $Y_i^{(j)}$, where i is the student (in our context an observation) and j is the question (a measurement).

Rasch modelling is one of the most popular strategies within IRT [13], [14]. It posits that there are *latent* variables, namely:

1. the ability or proficiency of student i, denoted α_i; and
2. the difficulty of question j, denoted β_j;

that determine the probability that student i answers question j correctly. The variables are latent in that we do not know them *a priori*.

In its simplest case, *i.e.,* a dichotomous response, we have response variables $Y_i^{(j)}$, which are Bernoulli random variables indicating a successful answer to a question, whose probabilities are modelled as

$$p_i^{(j)} = \mathbb{P}\left\{Y_i^{(j)} = 1\right\} = \frac{\exp(\alpha_i - \beta_j)}{1 + \exp(\alpha_i - \beta_j)}. \tag{1}$$

The *logistic* function above has an inverse called the *logit* function. Applying the logit to (1) gives

$$\text{logit}(p_i^{(j)}) = \log\left(p_i^{(j)}/(1 - p_i^{(j)})\right) = \alpha_i - \beta_j, \tag{2}$$

a linear relationship between the logit and the parameters.

Rasch modelling's enduring appeal within IRT [13], [14] arises because:

- it simplifies the relationships so that reasonable estimates can be made, even though we have only one instance of each student attempting each question;
- unlike the conventional statistical paradigm, where parameters are fit to data, and accepted or rejected based on the accuracy of the fit, in Rasch modelling the objective is to obtain "data" that fit the model, *i.e.,* the latent predictor variables; and
- the Rasch model embodies the *principle of invariant comparison*, in which (broadly speaking) the effect on the outcome of a question is separated into the affect of the respondent, and the question's difficulty.

The approach is not limited to modelling examinations, but can be applied to a set of observations such as we have. However, in traditional *dichotomous* Rasch analysis, each student answers each question one of two ways (correctly or incorrectly). Here, the naïve approach would be to consider each observation as a "student" with two questions (the HTTP and HTTPS measurements), but then we would have well above the number of students typically considered, blowing up the computational load for most algorithms. Moreover, this would be banal, as performance at this level of granularity is immaterial to us.

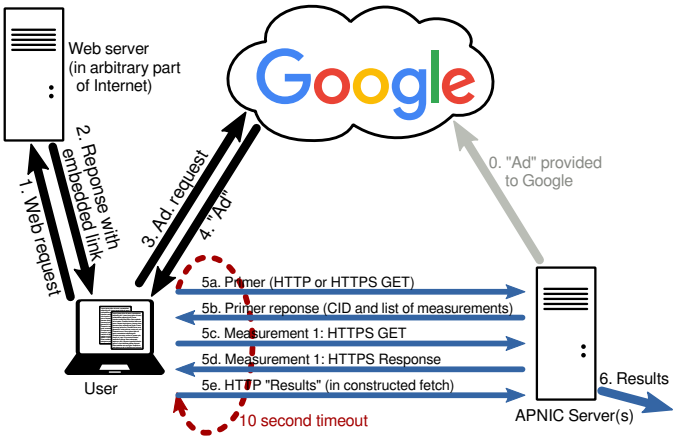


Fig. 1: *The observation process: numbers indicate sequence.*

In practice, we would like to group measurements into meaningful partitions, but we then depart from the standard dichotomous Rasch model.

There are at least two alternative approaches; we might think of these partition's subsets as either being comprised of:

1. a group of similar students, who have an underlying property in common (usually we assume members of the group have proficiencies that are random variables with a common mean and variance); or
2. a group of repeated measurements of a single "student" who corresponds to the particular subset, and the responses are now binomial random variables corresponding to the number of correct measurements within the subset.

The two assumptions lead naturally to different algorithms, and as the second is non-standard, we leave analysis its details until Section IV.

III. Experimental Method

A. Measurements

APNIC Labs uses web advertising to measure browser behaviour worldwide [12]. The advertisement is written in HTML5 and fetches multiple pixels in the various protocol exchanges under test (DNS, TCP/UDP, IP, TLS). The system is 100 lines of JavaScript, gzip compressed to 5kb of data, which is a small cost in web-page loading.

The process is illustrated in Figure 1. The observations start (at 5a) with a *primer* query initiated by the advertisement served to the user's browser via standard advertisement infrastructure. The primer query is an HTTP GET, and the body of the response is a set of measurements to be performed. Each measurement is a discrete URL with the unique client identity (CID) encoded in it, and is fetched under a ten-second timeout via an asynchronous JavaScript web fetch; on completion of a measurement, the time is recorded. On completion of all measurements, or the ten-second timer, a *result* web query is sent, which encodes the measurement results in the query argument as a sequence of labels, showing the time or 'null' if they did not complete inside the time limit.

The web logs show whether a primer/result pair was valid, and if so, we analyse the results. Observations without primer and result success are filtered from the sample. The goal in discarding these is to focus on the measurements with the highest signal to noise ratio — measurements without a valid pair indicate problems other than a failure of HTTPS, and hence don't add much information.

TABLE I: *Experiment duration, and number of observations. Analysis focuses on the 3.3 million experiments initiated with HTTP (with a subsequent HTTPS GET).*

Duration (days)	Unique client IDs (millions)	Valid responses (millions)	HTTPS init. (millions)	HTTP init. (millions)
25	192.5	132.4	129.1	3.3

The primary goal of these measurements was to collect information about ability to perform HTTPS. Google requires that advertisements placed over a TLS-secured session remain in TLS. Thus we could not recruit TLS users into a test of insecure web access. However, we were permitted to take an HTTP session and include fetches of web elements over TLS. Therefore, our observations measure HTTP users who were asked to fetch a web asset over TLS, thus detecting their ability to upgrade to TLS, which is not precisely a raw HTTPS access.

We focused on connections initiated over HTTP because this HTTP signal provides a "control." The priming process and the HTTP control measurement follow an identical connection path. Hence, if the observation is valid, the client has demonstrated the ability to perform an HTTP GET; therefore failures of subsequent HTTP GETs provide an indication of the "noise" in the system, *i.e.,* the baseline rate of random loss against which we should measure HTTPS connection failures.

The data were collected between the 10th of November and 4th of December, 2016. Table I shows the total set of client IDs, and the number of valid responses. A large number of connection attempts defaulted to initiating over HTTPS. Table I shows the decrease in the number of experiments as we progress through HTTPS to only HTTP initiations.

Initial exploratory analysis suggested that a signal existed, but at an intensity that could not be easily measured. The situation is analogous to experiments conducted on mice who are genetically modified to have cancer. We wished to measure factors that affected a situation with small probability, and so we inflate the probability of seeing the phenomena of interest. In our case, we focused on observations where the initial connection was HTTP, because these were the cases where failures of HTTPS were most often expected to occur.

As noted, it is a standard statistical approach to collect data in this way, but we must note that the observation is not representative of a "typical" Internet user. For instance, were we to measure a failure rate of 1% on these observations, this does not mean that the general population has a 1% failure rate. However, the question of interest here is not the absolute value of the failure rate, but whether HTTPS is "harder" than HTTP, and what factors affect the failure rate.

More formally, the main goal was to measure success/failure for sessions upgrading to TLS and to see if those sessions which could not upgrade to TLS were still successful on port

80. In other words: *"are there stranded users?"*

Google's infrastructure does not carry forward the referring site, and DHCP can reallocate IP addresses, so we cannot be certain that there were no repeats. However, the advertising infrastructure is intended to reach many discrete individuals, so the number of repeats should be very small. We also removed the small number of obvious duplicates from the data. There is some complexity in this process, resulting from apparent fetches from the same IP address that cannot be resolved due to the potential presence of middle-boxes such as Network Address Translators (NATs). We preserved entirely unique requests for the primer, but removed additional fetches without a new primer. As a result, we cannot claim that there are no duplicated observations, but they should be minimal.

B. Data Collected

The experiment logged all of the web fetches, using domain names directed to APNIC-managed DNS and web servers. We also captured the packet flow to relevant services: port 80, port 443, ICMP, DNS, as well as any fragmented IP state.

The combination of web logs, DNS logs, and packet captures allowed us to collate experiments by their IP address and identity in the DNS name, and as presented to the web. Thus we were able to derive the exact sequences of events in any observation.

In the case of this experiment, the data were processed into the form of a series of flags indicating (1) the success or failure of each stage, and (2) whether the measurement succeeded within a timeout. The delays were recorded in each case up to 120 seconds, but for our purposes we recorded success if the measurement completed within a timeout of 10 seconds.

In the data analysed, unique client IDs were assigned to anonymise the data. We used code which harvests system entropy and time, to obtain probably unique (modulo birthday paradox) non-sorted 96-bit numbers. We then mapped them into hex (see [25], for the code that was embedded in the NGINX [26] web server).

C. Classification of Covariates

The secondary goal here was to identify the qualities behind the quantities: *i.e.,* can we understand these users in terms of browser type, ISP, economy, or operating system, in order to identify specific problem causes? In practice, this is important because the goal behind APNIC's participation in such experiments goes beyond simply finding problems. Ideally, the experiment should also help develop strategies to remediate any problems found.

For the purposes of the analysis, a set of qualities was identified, which we felt were simple, easy to reproduce by other people, and provided useful groupings for understanding causes. These qualities were:

- country,
- region (based on United Nations sub-regions [27]),
- origin Autonomous System Number (ASN),
- browser, and
- Operating System (OS).

We used the daily BGP table collected at AS4608 to map IP addresses to origin ASN. There are well-known problems in such a mapping. However, those problems are most prevalent in infrastructure addresses, and we measured "eye-balls" here; inter-AS links do not browse [28].

Likewise, mapping of eye-balls to geographic locations is more accurate [28] than mapping arbitrary IP addresses to geographic locations. In this paper, we used MaxMind [29] data to geolocate the IP addresses, but only at the country/region levels, and so expected a reasonably low error rate.

We also logged each client's user-agent string, which provides details of the client's browser, OS, and device. To collect and parse the information we used the Python *uabrowser* library [30]. It is known that the user-agent string is spoofed in some cases, for instance the ToR browser bundle does so by default (*e.g.,* it pretends to be running on Windows, regardless of the underlying OS). However, there is no easy way to avoid this problem at present, and it remains a caveat on the browser- and OS-level results.

We also considered categorising the client's device-type, but this was too noisy to be useful at this stage, due to the large number of uniquely identified device types by vendor and version-string.

For each of these categories, we collated them into a series of unique values, and then used a one-way random relabelling to anonymise the categories. There might be enough data to perform some act of deanonymisation, to obtain values for some categories, However, it is important to note that this level of blinding was not intended for the protection of individual privacy (already protected through the client ID anonymisation, and unlikely to be compromised by the additional coarse-grained categorisations). Rather it was intended to allow the statistical analysis to proceed, unbiased by preconceived notions of the likely results.

D. Ethical Concerns

Google's advertising infrastructure was used here, and so we reviewed compliance with Google and APNIC lawyers, and complied with Google's legal restrictions on the measurements. In particular, these excluded use of Personally Identifying Information (PII), which in any case we did not require or want. End-user IP addresses were only used for ASN and regional classification, and were then anonymised via a one-way-mapping.

As in many Internet measurement experiments the nature of the measurement technique precluded voluntary recruitment. However, we strictly followed any suggestion that the users wished to opt out of such studies. For instance, end users who had enabled 'do not track', who had disabled JavaScript, or who ran ad-blocking software were not recruited. Also, as far as possible, no users were repeatedly asked to run the experiment, in order to place minimum load on any one user.

These measurements were also less contentious than others that have applied similar cross-origin requests *e.g.,* [22]. TLS is used ubiquitously for banking, login, end-user tracking by less responsible advertisers, "bread crumbs" and web-site

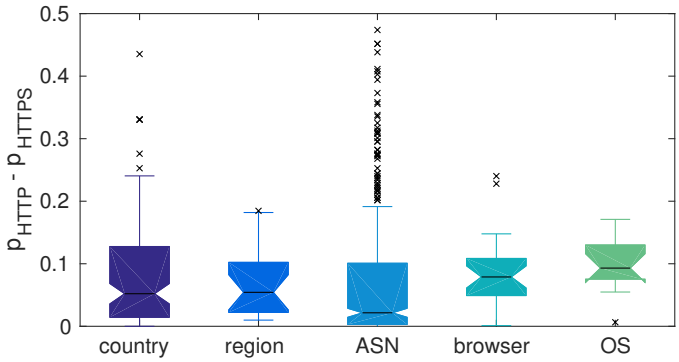


Fig. 2: *Box and whisker plot of difference $p_{HTTP} - p_{HTTPS}$ by covariate, showing the interquartile range (shaded region) the confidence interval for the estimate of the median (notch), Tukey's 1.5 IQR, (whiskers) and outliers (crosses). Note that the lower quartile is always positive, as are all whiskers except those for ASN, suggesting that HTTPS* is *harder than HTTP. Also, if covariates had no effect on the result, these should all have similar interquartile range and median, the differences therefore suggest some structure.*

logistics. Evidence suggests that the rate of TLS in the public web is high (above 50% [7]) and very likely significantly higher given the age of that study, and that it has been rapidly increasing in recent years [9], [10]. Therefore, the simple presence of a request to fetch a web asset over TLS does not represent a high-risk activity.

Moreover, the measurement site to which the advertisement redirected requests is innocuous, belonging to a regional address registry (APNIC), so we were not able to discern any reasonable risk to participants from such a connection.

In this experiment, those researchers not employed by APNIC were exposed only to anonymised data, except for those statistics reported here.

IV. Analysis

In this section, we discuss the results of the analyses. We will start with "broad brush" simple hypothesis tests, then focus on those same tests, applied to country, region, ASN, OS and browser. This will be followed by a more comprehensive Rasch model, which analyses the data as a whole.

Figure 2 shows a box and whisker plot [31], [32] of the differences organised by the various predictors. Note that the lower quartile is always positive, as are all whiskers except those for ASN, suggesting that HTTPS *is* harder than HTTP. Our task is to determine whether this effect is statistically significant.

Also, if covariates had no effect on the result, these should all have similar interquartile range and median, the differences therefore suggest some structure.

A. Standard Statistical Tests

We applied McNemar's test with a significance level of $\alpha = 0.05$, applying the appropriate Bonferroni corrections when conducting a set of multiple tests (*i.e.,* we used significance α/n for a family of n tests). Note that in some cases, *e.g.,* when we were testing against ASN, n was quite large, and so the actual threshold was very small. Less conservative corrections exist (for instance the Sidak or Holm-Bonferroni) but the results here are conclusive without needing the extra power gained through these more accurate corrections.

TABLE II: *Statistical tests applied to the whole dataset. Note that very small p-values are reported via a bound.*

Test	p-value	Accept/Reject
Fisher	$< 2 \times 10^{-16}$	Reject null
McNemar	$< 2 \times 10^{-16}$	Reject null

The results, shown in Table II, for the test applied to the whole dataset was a p-value less than 10^{-16} strongly supporting a difference in the two measurements. This finding must be qualified: although the two measurements are matched they occur in order, and hence, there may be some effect on the second measurement resulting from the state created by the first. So we must understand that this experiment concerns lifting a connection up from HTTP to HTTPS, not an arbitrary HTTPS connection (see the detailed notes in Section III).

It is important, also, to verify that this is not caused by some confounding effect of the covariates. If the covariates were truly irrelevant, we would expect that interquartiles and medians should be the same (within the ranges of natural variation shown by the confidence in intervals in the case of the median), and hence Figure 2 provides evidence that the covariates are important.

Therefore we now consider what part the covariates (country, region, ASN, OS and browser) play. We chose, at least initially, to be conservative by only analysing groupings with at least 500 observations. It is quite possible that smaller groupings would have been amenable to analysis, but we had no need (here) to describe the relationships between all of the rarer groupings, as our goal was to ascertain whether the overall result was supported on a finer level of granularity.

The number 500 was chosen through an initial exploratory analysis, which noted that some of the probabilities in question were quite close to 1, and hence statistical rules of thumb required a moderately large number of observations. As we did not know exactly what these probabilities were *a priori*, we chose a conservative lower bound. We also found, as Table III shows, that excluding the groups with a small number of observations excluded only a small percentage of the data.

The results can be seen in Figure 3, which shows histograms of the distributions of p-values over the set of tests grouped by the various categorical covariates described above. The important fact to note is that most of the p-values are small. We cannot see (at the resolution of the plot) whether the p-values fall below the threshold, so Table III summarises the tests, showing that in a large proportion of the cases we should reject the null-hypotheses. Thus we have significant evidence for a difference in most of the groupings.

Interestingly, ASN is the grouping with the lowest proportion of rejected null-hypotheses, while we might have expected that ASN would have a larger effect on the network aspects of the problem. However, remember the large Bonferroni correction in this case, which leads to a very conservative test.

Hypothesis testing could be expanded here in several ways.

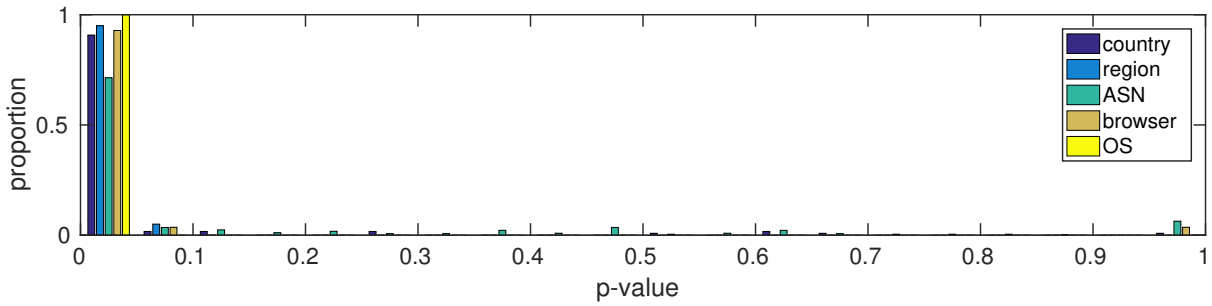


Fig. 3: *The distributions of p-values for the McNemar tests, applied across country, region, ASN, browser and OS. The distributions in all cases vary dramatically from a uniform distribution, with values heavily skewed in the direction of $p = 0$.*

TABLE III: *Hypothesis tests summaries for different covariates. $\tilde{N}$ is the number of groups left after excluding those with fewer than 500 observations. The "% of data" is that retained by this filtering. And the final column reports the proportion of McNemar tests for which we reject the null hypothesis over the $\tilde{N}$ groups.*

covariate	$\tilde{N}$	% of data	McNemar
country	119	99.6	0.840
region	20	100.0	0.950
ASN	458	93.1	0.555
browser	28	99.9	0.929
OS	14	100.0	1.000

Multiple-comparisons could be applied, for instance, to test differences between countries or some other covariate. However, in doing so there would be $O(n^2)$ comparisons for n countries, and these hypothesis tests are not all independent of each other, complicating the test procedure greatly. Moreover, much of this theory has been developed in domains where the each measurement requires a physical or social experiment, and therefore it seeks to make best use of a limited set of costly measurements. We have many measurements, and so these refinements are not needed. Instead, in our next step we opt to apply an approach called *Rasch analysis*.

B. Rasch Analysis

The disadvantage of the previous tests is they provide only a yes/no answer (or really a yes/maybe answer), while we would like, for instance, to be able to say how large the difference is. Here we use Rasch modelling to perform this analysis, but as we are not interested in the per-observation performance we use a grouping strategy. We start by defining G_k to be the kth group of observations determined by a covariate. We consider here two modelling approaches.

The first model still follows (2), but now we assume that the proficiency of each student is distributed as $\alpha_i \sim N(\lambda_k, \sigma^2)$, for $i \in G_k$, where λ_k is the group mean proficiency, and σ^2 is the common standard deviation within groups. The task is then to estimate λ_k and σ^2. The careful reader will note the additional assumptions introduced by this model.

The second approach takes a simpler model, that

$$\text{logit}(p_i^{(j)}) = \alpha_k - \beta_j. \tag{3}$$

for $i \in G_k$, We now only estimate a group proficiency α_k, not individual proficiencies. This has the disadvantage that it might not be able to fit the data as accurately, but it frees from distributional assumptions.

The former approach has been developed further, in that exact results are known, and there are off-the-shelf solvers using *Marginal Maximum Likelihood Estimation* (MMLE). We use IRTm [33], [34], a Matlab toolbox allowing quite general models to be estimated. Apart from its additional assumptions, in MMLE each categorical variable with m categories is deconstructed into m binary variables, each an indicator for one possible state of the original variable. For instance, the 119 countries in our data result in constructing a covariate vector consisting of 119 binary elements, leading to an estimation procedure taking considerable memory and time.

The results are illustrated below, in conjunction with those of the second approach, in which we assume each group consists of a set of repeated measurements. However, the standard Binomial Rasch models assumes each measurement is repeated a fixed number of times. For instance, in partial-credit Rasch models [35], a student may obtain some proportion of the marks for a question, but each student answers the same question, with the same total possible marks. But in our groupings, the number of "total marks" would vary, depending on the number of client observations that fall into the group. This case does not appear to have been treated in the literature, and hence we wrote our own Alternating Least Squares (ALS) algorithm (also in Matlab) to estimate the parameters.

The algorithm alternates between fitting the α_k and the β_j values, keeping the other parameters fixed. It also needs an additional fixed point of reference (because the variables α_k and β_j are not otherwise uniquely determined), which we fix, without loss of generality, by $E[\alpha_k] = 0$.

We assessed the two approaches in this (somewhat non-standard) application by comparing computation times, and Root-Mean-Square (RMS) fitting errors, as shown in Figure 4. All computations were made on an 8 core, Intel i7-6900K 3.2 GHz, running Linux Mint 18, and Matlab R1016b (the largest case for the MMLE algorithm did not complete within 24 hours and so is excluded), The ALS algorithm is orders of magnitude more accurate and faster, and so in what follows we focus on the ALS approach. Note also that the estimation errors in ALS reach a maximum in the order of 5%, which is reasonable given the problem of interest.

Ideally, we could group by all covariates at once. However,

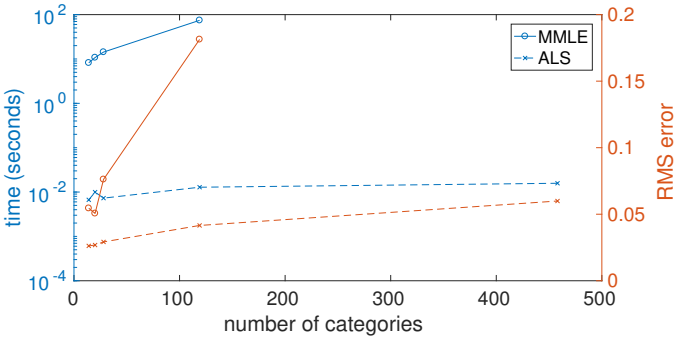


Fig. 4: *Computation times (blue) and RMS errors (red) for the two approaches to Rasch modelling.*

this results a *very* large number of covariates in the MMLE approach, while in the ALS approach, we end up with very few observations in many of the bins, due to the combinatorial number of bins. Thus we analyse each of the categories as separate groupings. As before, we consider only groupings with at least 500 observations.

The first detail to consider is the β_j values, indicating the difficulty in completing the two measurements (HTTP, and HTTPS). The estimated values are shown in Table IV, along with their difference. Larger values indicate additional difficulty with a measurement. The positive values of the difference indicate additional difficulty in the HTTPS measurements compared to HTTP. From all points of view, HTTPS is more difficult than HTTP.

We also see some consistency, namely, the differences in β_j are similar for location (country, region and ASN), and for end-point software (OS and browser), as you might expect. Notably the former group seems to have a larger impact on success than the latter, so it appears that while a client's device is important, the location from which one accesses the Internet is more important.

The second set of parameters to examine are the α_i values, namely, the ability or proficiency of a particular covariate group to perform any of the measurements, large values being better. Figure 5 shows the distribution of α_i values for the region, OS, and browser covariates. We see that they might be coarsely considered to follow a Normal distribution. The data by OS fit this assumption least well, but remember that there are only 14 values here, and we expect to see some natural variation here, because of measurement noise.

Note that we do not draw, from these values, inferences about the particular quality of HTTPS in a particular country (or other grouping). The α_k variables record the ability of a group to perform both HTTP and HTTPS measurement. This parameter separates out the "noise" inherited from the quality of Internet connections through a particular country from the HTTP v HTTPS question.

However, we also see outliers, here defined as those values that fall more than 1.96 times the standard deviation from the mean, *i.e.,* outside the 95th percentiles. These are not extreme outliers, but there may be some interest in these, so we have reversed the mapping (for these outliers only).

TABLE IV: *ALS estimates of Rasch "difficulty" parameters with different groupings. Larger values indicate a smaller chance of measurement success. Note the increase in difficulty for HTTPS.*

	country	region	ASN	browser	OS
β_{HTTP}	-5.26	-4.91	-6.07	-3.94	-3.99
β_{HTTPS}	-2.92	-2.74	-3.80	-2.29	-2.12
Difference	2.34	2.16	2.27	1.65	1.86

- **country**: five positive outliers: Suriname, Macau, Cyprus, Latvia, Korea; and one negative: Macedonia.
- **region**: no outliers.
- **ASN**: there is a list of 22 positive outliers, but only one negative: AS58539 (listed as China Telecom).
- **browser**: positive outlier: Amazon Silk and no negatives.
- **OS**: one positive outlier: ChromeOS and no negatives.

Some of these might be slightly surprising – for instance, many may not have expect Suriname to be in the list of positive outliers. However, it should be remembered that the measurement methodology filters participants who successfully complete the primer and results query successfully. So this result really says that, those who have a good connection, have a very good connection, *i.e.,* , if they complete the primer and result, they are very likely to be able to complete the other measurements.

Similarly, the positive outliers ChromeOS and Amazon Silk (the Kindle Fire's browser) are perhaps indications of consistency amongst all such devices, because of the stronger constraints on these devices. For instance, Amazon Silk routes requests through remote proxy servers powered by Amazon EC2, which provide high-performance connection speeds and computing power not normally available to a mobile form factor, and apparently improve the consistency of responses.

Hence, though these outliers may be interesting, the underlying point is not that any particular location or device group has a given α_k, so much as the parameter allows us to disentangle these effects from those of the two measurements (HTTP and HTTPS), and thus see the latter in isolation.

In summary, there are two main conclusions to be drawn.

1. The measurements show that there is significantly more difficulty in performing HTTPS than HTTP measurements. The difference is often small, necessitating some extra care in order to determine whether the difference is significant.
2. There are country, OS, and browser differences, mainly important through a small set that exhibits more extreme variations from the norm.

One last important insight is that the dependence on the covariates indicates that the results are not an artefact of APNIC's measurement infrastructure, as that would remove dependency on point of origin effects.

V. Conclusion and Further Work

Using a large set of measurements (data provided at bandicoot.maths.adelaide.edu.au/HTTPS/), and detailed statistical modelling we have shown that a small cohort of users in the real world will be adversely affected if HTTPS is adopted universally. That cohort is not a large proportion of Internet users, but those users deserve our attention.

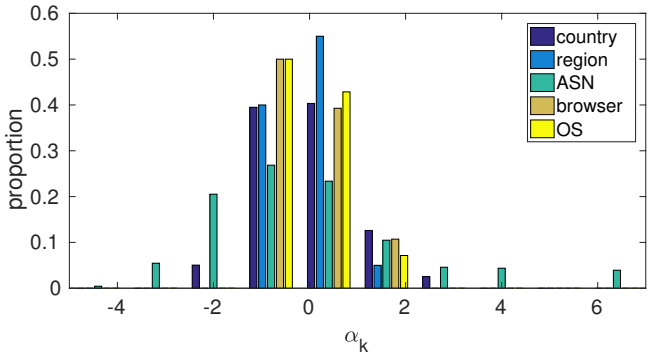


Fig. 5: *Histograms of α_k values for ALS algorithm. Positive values indicate a favourable probability of measurement success.*

We have categorised measurements by country and region, their provider (origin ASN), browser and operating system, and shown that all of these factors affect a client's facility with HTTPS. The range of factors points to a range of causes for the blockages. The browser/OS combination suggests a technological problem, but the other covariates suggest problems based in the network near the clients.

In the future, we plan to further investigate, and use the details of the analysis with extensions to understand better correlations in covariates, to help focus efforts onto relevant development to mitigate the problem.

The use of careful statistical methods was vital in this study. The underlying signal is weak, and hence required "amplification" and careful analysis so as to be able to make confident statements.

Acknowledgements

We would like to thank the Australian Research Council for funding through the Centre of Excellence for Mathematical & Statistical Frontiers (ACEMS), and grant DP110103505.

The Javascript code used by APNIC originates in a library written by Emile Aben, RIPE-NCC.

APNIC Labs has received support and in-kind assistance from Google, ICANN, RIPE-NCC and ISC in its experiments.

References

[1] "HTTPS everywhere," on-line: downloaded April 24th, 2017, https://www.eff.org/https-everywhere.

[2] "HTTPS everywhere," Google I/O, https://docs.google.com/presentation/d/15H8Sj-Zol1tcum0CSylhmXns5r7cvNFtzYrcwAzkTjM/edit#slide=id.g12f3ee71d_10.

[3] "HTTPS as a ranking signal," https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html, August 2016.

[4] "The HTTPS-Only standard," on-line: downloaded April 24th, 2017, https://https.cio.gov/.

[5] A. Goldberg, R. Buff, and A. Schmitt, "A comparison of HTTP and HTTPS performance," in *CMG98*, 1998, http://www.cs.nyu.edu/artg/research/comparison/comparison.html.

[6] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of tls web servers," *ACM Trans. Comput. Syst.*, vol. 24, no. 1, pp. 39–69, Feb. 2006. [Online]. Available: http://doi.acm.org/10.1145/1124153.1124155

[7] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the 'S' in HTTPS," in *10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, New York, NY, USA, 2014, pp. 133–140. [Online]. Available: http://doi.acm.org/10.1145/2674005.2674991

[8] D. Beaver, "HTTP2 expression of interest," on-line: downloaded April 24th, 2017, July 2012, http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0251.html.

[9] "HTTPS usage statistics on top websites," on-line: downloaded April 24th, 2017, https://statoperator.com/research/https-usage-statistics-on-top-websites/.

[10] "HTTPS usage," on-line: downloaded April 24th, 2017, https://www.google.com/transparencyreport/https/metrics/?hl=en.

[11] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the web," in *USENIX Securit*, 2017.

[12] G. Michaelson and G. Huston, "Experience with large-scale end user measurement techniques," in *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*. IEEE, 2014, pp. 1–5.

[13] B. Wright and M. Mok, *Introduction to Rasch Measurement: Theory, Models, and Applications*. Journal of Applied Measurement, 2004, ch. An Overview of the Family of Rasch Measurement Models, jampress.org/irmch1.pdf.

[14] B. D. Wright, "Solving measurement problems with the Rasch model," *Journal of Educational Measurement*, vol. 14, no. 2, pp. 97–116, 1977. [Online]. Available: http://www.jstor.org/stable/1434010

[15] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *ACM Sigcomm Internet Measurement Conference*, 2013, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504755

[16] B. Laurie and C. Doctorow, "Secure the internet," *Nature*, vol. 491, pp. 325–6, 2012.

[17] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the HTTPS protocol," *IEEE Security Privacy*, vol. 7, no. 1, pp. 78–81, Jan 2009.

[18] D. Ranathunga, M. Roughan, H. Nguyen, P. Kernick, and N. Falkner, "Case studies of SCADA firewall configurations and the implications for best practices," *IEEE Transactions on Network and Service Management*, 2016, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7529047&isnumber=5699970.

[19] "SSL/TLS - typical problems and how to debug them," on-line: downloaded April 24th, 2017, https://maulwuff.de/research/ssl-debugging.html.

[20] S. Fahl, Y. Acar, H. Perl, and M. Smith, "Why Eve and Mallory (also) love webmasters: A study on the root causes of SSL misconfigurations," in *ASIA CCS*, 2014.

[21] M. Casado and M. J. Freedman, "Peering through the shroud: The effect of edge opacity on IP-based client identification," in *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, ser. NSDI'07, 2007, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973430.1973443

[22] S. Burnett and N. Feamster, "Encore: Lightweight measurement of web censorship with cross-origin requests," in *ACM SIGCOMM*, 2015, pp. 653–667. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787485

[23] J. P. Shaffer, "Multiple hypothesis testing," *Annu.Rev.Psychol.*, vol. 46, pp. 561–584, 1995.

[24] A. Agresti, *Categorical Data Analysis*, 2nd ed. Wiley, 2002.

[25] "ngx_txid," https://github.com/APNIC-Labs/ngx_txid, accessed May 16th, 2017.

[26] "nginx," http://nginx.org/, accessed May 16th, 2017.

[27] "Standard country or area codes for statistical use (M49)," https://unstats.un.org/unsd/methodology/m49/, accessed May 16th, 2017.

[28] A. H. Rasti, N. Magharei, R. Rejaie, and W. Willinger, "Eyeball ASes: From geography to connectivity," in *ACM Sigcomm IMC*, Melbourne, Australia, 2010.

[29] "GeoIP databases & services: Industry leading IP intelligence," https://www.maxmind.com/en/geoip2-services-and-databases, accessed May 16th, 2017.

[30] "A Python implementation of the UA parser," https://github.com/ua-parser/uap-python, accessed May 16th, 2017.

[31] R. McGill, J. W. Tukey, and W. A. Larsen, "Variations of box plots," *The American Statistician*, vol. 32, no. 1, pp. 12–16, 1978. [Online]. Available: http://www.jstor.org/stable/2683468

[32] H. Wickham and L. Stryjewski, "40 years of boxplots," had.co.nz, Tech. Rep., 2012, http://vita.had.co.nz/papers/boxplots.html.

[33] J. Braeken and F. Tuerlinckx, "Investigating latent constructs with item response models: a MATLAB IRTm toolbox," *Behavior Research Methods*, vol. 414, no. 4, pp. 1127–37, 2009.

[34] "IRTm," https://ppw.kuleuven.be/okp/software/irtm/.

[35] G. M. Masters, "A Rasch model for partial credit scoring," *Psychometrika*, vol. 47, no. 2, pp. 149–174, 1982.

PathFinder: Capturing DDoS Traffic Footprints on the Internet

Lumin Shi*, Mingwei Zhang*, Jun Li*, Peter Reiher†
* University of Oregon
{luminshi, mingwei, lijun}@cs.uoregon.edu
† University of California, Los Angeles
reiher@cs.ucla.edu

***Abstract*—While distributed denial-of-service (DDoS) attacks are easy to launch and are becoming more damaging, the defense against DDoS attacks often suffers from the lack of relevant knowledge of the DDoS traffic, including the paths the DDoS traffic has used, the source addresses (spoofed or not) that appear along each path, and the amount of traffic per path or per source. Though IP traceback and path inference approaches could be considered, they are either expensive and hard to deploy or inaccurate. We propose PathFinder, a service that a DDoS defense system can use to obtain the footprints of the DDoS traffic to the victim as is. It introduces a PFTrie data structure with multiple design features to log traffic at line rate, and is easy to implement and deploy on today's Internet. We show that PathFinder can significantly improve the efficacy of a DDoS defense system, while PathFinder itself is fast and has a manageable overhead.**

***Index Terms*—distributed denial-of-service; DDoS; traffic footprint; autonomous system (AS); PFTrie**

I. Introduction

Today's Internet is vulnerable to distributed denial-of-service (DDoS) attacks. During a DDoS attack, an attacker controls many compromised machines to flood the victim with unwanted traffic in order to exhaust the network or computational resources of the victim. DDoS attacks have become more frequent and damaging to many network services [1]. For example, a recent large-scale DDoS attack on Dyn [2] disabled its domain name service, and crippled many major web services that relied on it such as Twitter, Netflix, PayPal, and over fifty others for hours.

While many DDoS defense systems have been proposed, a primary challenge in effectively defending against DDoS attacks is that a DDoS defense system usually has little knowledge regarding which paths DDoS traffic has traveled along, how much traffic traveled along each path, and also which source addresses or prefixes of the DDoS traffic are associated with each path. Such information about the DDoS traffic, which we collectively call the **footprints** of the DDoS traffic, if available, can enable a DDoS defense system to most effectively handle the DDoS attack. It will become more informed regarding where to deploy DDoS traffic filters or take other defense actions; for example, it may learn which autonomous systems (ASes) or AS paths have seen a large amount of traffic to the victim, thus deploying filters there. It can also know better which source addresses (or more likely the source IP prefixes, for scalability) to filter in the case of source-based filtering. And it could also conduct traffic pattern analysis if the footprints are continuously provided.

Various approaches to obtaining such information could be considered, including numerous IP traceback approaches and path inference methods that aim to address the asymmetric nature of Internet paths and ascertain the paths traveled by DDoS packets to reach the victim. Unfortunately, as we will discuss in more detail in Sec. II, these approaches have serious drawbacks. For example, the path inference methods are often inaccurate, and IP traceback approaches introduce significant changes to router hardware or software, rely on inter-AS collaboration, and need routers on the Internet to *constantly* monitor the traffic. These approaches are also not well-equipped to provide other footprint information, such as total or per-source bandwidth consumption information or the addresses or prefixes of DDoS sources.

We therefore introduce the **PathFinder** system as a service for DDoS defense systems. Upon request from a DDoS defense system on behalf of a DDoS victim, PathFinder can gather and provide the footprints of the traffic to the victim. We make the following contributions:

- PathFinder consists of an architecture that is easy to implement and deploy on today's Internet. Every AS can join PathFinder without reliance on other ASes, and it employs an *on demand* service model with low overhead.
- We design the setup and operations of each component of the architecture while considering a series of real-world factors and the high speed and large scale of DDoS traffic.
- We design a new data structure called **PFTrie** that supports fast and easy storage and retrieval of traffic footprint information. And we also design a set of PFTrie optimization methods.
- We show the benefits of using PathFinder for DDoS defense, and we further evaluate PathFinder to show it is fast and has a manageable overhead.

ISBN 978-3-903176-08-9

This project is in part the result of funding provided by the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00204. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security or the US Government.

An apparent issue here is IP spoofing. The DDoS traffic could carry spoofed IP source addresses. However, we note that regardless whether the IP source addresses of DDoS traffic are spoofed or not, PathFinder will still discover the correct set of paths that DDoS traffic take to reach the victim, thus enabling a DDoS defense system to use the path information, together with other traffic footprints information, to block the DDoS traffic *en route* accordingly. On the other hand, if a DDoS defense system needs to filter DDoS traffic based on the source addresses of DDoS traffic, the DDoS defense itself needs to handle the IP spoofing separately, which is out of the scope of PathFinder.

The rest of this paper is organized as follows. We first discuss related work in Sec. II, followed by an overview of PathFinder in Sec. III. We then describe individual components of PathFinder, including the PathFinder monitor in Sec. IV, the PFTrie data structure for traffic logging in Sec. V, and the PathFinder proxy in Sec. VI. We evaluate PathFinder in Sec. VII, discuss some open issues in Sec. VIII, and conclude the paper in Sec. IX.

II. Related Work

A. IP Traceback

IP traceback, first introduced in [3], allows a victim to trace the source of an IP packet it has received and reconstruct the router-level path taken by the packet, even if the source address of the packet is spoofed. While many IP traceback solutions have been proposed [4], marking and logging are the two most well-developed approaches.

In a marking approach, such as those described in [5], [6], [7], when a router along a path forwards a packet to the victim, the router marks the packet with its own IP address (or its hashed result) or an edge that the packet has traversed, typically using some unused fields in the IP header of the packet. When the victim receives *enough* marked packets, even if routers *en route* mark packets with certain probability rather than all the time, the victim can then reconstruct the paths of these packets (assuming the paths are stable).

In a logging approach such as [8], [9], as a router along a path forwards a packet to the victim, instead of marking the packet, the router uses some data structure (e.g., a Bloom filter) to store the digest of the packet (rather than the packet itself in order to save space), enabling it to later determine whether it has seen the packet. When the victim wants to trace a packet, it can query its upstream routers, asking whether they have seen the packet. Similarly, a router that has seen the packet can query its own neighboring routers about the packet, and so on. Eventually the victim can reconstruct the packet's path using an ordered list of routers that have seen the packet.

PathFinder has advantages over existing IP traceback approaches in the following respects:

- *Operation*: PathFinder is also essentially a logging approach, but while the previous logging-based IP traceback approaches try to trace the path(s) of any packets (such as packets from a specific source) that a victim has received, PathFinder aims to discover the paths of *upcoming* packets (often all upcoming packets within a time window) toward a victim when requested. So, while existing IP traceback approaches record packet information or mark packets constantly, PathFinder is an on-demand service and PathFinder monitors will only record traffic information when requested.
- *Overhead*: Because it operates on demand, PathFinder incurs much less operational overhead than existing IP traceback approaches. In addition, packet marking approaches will modify packets before forwarding them, which will introduce delays in processing packets and could downgrade the network throughput significantly, especially when dealing with a high-bandwidth link.
- *Accuracy*: The accuracy of the marking approaches depends on how many marked packets the victim can receive to reconstruct the paths of packets. The accuracy can suffer if the victim cannot receive enough marked packets, such as when its inbound link is congested with DDoS traffic. The existing logging approaches (and also PathFinder), on the other hand, as long as their monitoring mechanism can process packet headers at line speed, can log packet information with little loss and thus reach a high accuracy in tracing packets.
- *Deployability*: Existing IP traceback approaches face obstacles for deployment: Whether based on marking or logging techniques, they introduce significant hardware or software changes to routers, and also require inter-AS collaboration. PathFinder instead introduces few changes to routers, and PathFinder-participating ASes talk directly with a PathFinder proxy and do not need to communicate with each other.

B. Path Inference

Researchers have studied how to infer the path between two end points on the Internet. Without assuming any control over the network infrastructure or access to end points, research in [10] investigated how to leverage Border Gateway Protocol (BGP) tables collected from multiple vantage points to infer the AS path between any two end points on the Internet. Also, via the probing from multiple vantage points and the IP timestamp and record route options, research in [11] proposed a "reverse traceroute" to allow a user to infer the path from a remote end point to the user, without accessing the remote end point. Inference-based approaches do not require changes to network equipment and are easy to deploy, however, in general they are subject to some degree of inaccuracy (e.g., the accuracy from the research in [10] is 70-88%). Moreover, since they are not based on watching traffic in real time, they will not be able to report the bandwidth consumption and other traffic-related information associated with the path.

III. PathFinder Overview

A. PathFinder as a Service for DDoS Defense

We design PathFinder as a service for DDoS defense. Upon request from a DDoS defense system, PathFinder can provide the footprints of all the traffic toward a victim that each participating AS has witnessed. Note that PathFinder does

not distinguish DDoS traffic from the legitimate traffic, which PathFinder assumes to be the job of the DDoS defense. The footprints include:

- all the AS paths taken by the traffic;
- if requested, the source IP addresses or prefixes of the traffic; *and*
- if requested, the amount of traffic per source address, or per source prefix, or per AS *en route*, or other information about the traffic.

When requesting the PathFinder service, a DDoS defense can specify to PathFinder a set of parameters regarding the traffic footprints, including:

- the destination address of the victim, which could be an IP address or prefix, or an IP address plus a port number;
- the length of time for which to collect the traffic footprints (typically during the DDoS attack);
- from which ASes (if not all the ASes supporting PathFinder) to collect the traffic footprints;
- whether to collect the source addresses or prefixes of the traffic, and if so, the prefix granularity (e.g., /24 is to learn all the /24 source prefixes; /32 is to learn all the /32 source prefixes, i.e., all the source IP addresses); *and*
- whether to collect the bandwidth consumption of the traffic, and if so, the granularity of the bandwidth information (per source address, per source prefix, or per AS *en route*) and the unit (packets per second or bits per second).

B. Architecture

PathFinder is a log-based system that enables a client (which is DDoS defense in this paper) to learn the AS paths, sources, bandwidth consumption, and other information of the traffic toward a DDoS victim, i.e., the footprints of the traffic. As we will show in the following, it is easy to deploy as it requires minimal reconfiguration of routers; it is scalable as it will continue to perform well if there is more traffic from more sources or if more ASes support PathFinder; and it is accurate, fast, and efficient in providing the traffic information.

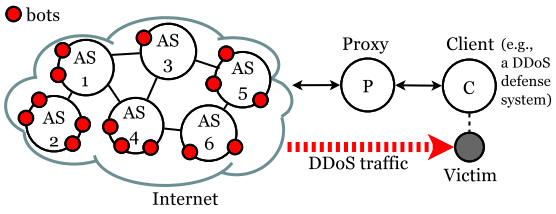


Fig. 1: **PathFinder architecture.**

Fig. 1 shows the high-level architecture of PathFinder. It consists of three types of entities:

- **PathFinder clients** who interact with their proxy to request the PathFinder service, and retrieve from their proxy the footprints of the inbound traffic to a DDoS victim;
- **PathFinder proxies** which (1) pass their clients' requests (including all parameters described in Section III-A) to all participating ASes—actually their PathFinder monitors; (2) receive and process from these ASes the PathFinder logs, which we call **PFLogs**, to derive the DDoS traffic footprints; and (3) return the footprints to their client; *and*
- **PathFinder monitors** at all participating AS which, according to the request from a proxy, (1) process the traffic that their AS originates or forwards towards the victim specified in the request; (2) generate PFLogs of the traffic, which record the AS path, source addresses (if requested), and amount (if requested) of the traffic; and (3) return the PFLogs to the proxy. Note that monitors from different ASes do not need to interact with each other, thus not introducing into the architecture any reliance on inter-AS collaboration.

IV. PathFinder Monitor

A. Addressing Design Requirements

Foremost, the monitor at each PathFinder-participating AS faces the following two design requirements. First, the monitor needs to consult routers within the AS to learn the AS path from the AS to the victim. Second, it also needs to access the traffic toward the victim in order to record their source addresses and/or amount, if requested. As an AS can have a complicated topology with inter-connected border routers and inside routers, some routers may not be on any path toward the victim at all and some may be on the same path. To meet both requirements, for every path of the traffic to the victim, the monitor must be able to talk with at least one router that is on the path, in order to learn its AS path to the victim or access its traffic to the victim. For the former (to learn its AS path), as every border router on the Internet runs BGP and maintains a Routing Information Base (RIB), the monitor can query the RIB at the router. Note that BGP router vendors such as Cisco [12] and Juniper [13] all support the query of AS paths. For the latter (to access traffic), the monitor needs to receive a copy of the traffic by applying traffic mirroring or tapping techniques (we rule out the possible hardware telemetry support from routers; although they produce traffic records such as those in NetFlow or IPFIX format, the records are only exported periodically, often with a long interval).

The monitor may further face a third requirement if it needs to produce PFLogs with source information or also the traffic amount records. There may be a huge amount of traffic from many distinct sources toward the victim, especially if the victim is currently under a severe DDoS attack, thus making it challenging for the monitor to record all the sources and their corresponding bandwidth consumption at a high speed. The most obvious solution is to use a digest-oriented data structure such as a Bloom filter or hash table. However, while the monitor can use a Bloom filter to easily answer whether it has seen an IP address or prefix or not, it is not good at recording which specific source IP addresses or prefixes it has seen. A hash table is better, but it can only output whatever is stored in itself as is; it is not flexible in processing or aggregating IP address and prefix information, thus scaling poorly when the logs are of a huge size. We therefore design a new, trie-based data structure called **PFTrie** to facilitate the recording and transmission of PFLogs, which we detail in Section V.

B. Setup

A PathFinder-participating AS needs to set up its PathFinder monitor and its working environment as follows. First, the monitor needs to set up the traffic mirroring or tapping with every border router from which it will need to obtain traffic *in real time*. To do so, given the autonomy of ASes, each AS may adopt a different procedure that it prefers. For a small AS without many routers, it can physically wire the monitor with every router for wiretapping (Fig. 2a shows an example). For a large AS with many routers over a large geographic region, we assume it can first learn which routers the AS has and then use virtual circuits for traffic mirroring with the routers [14], [15]. Further, a large AS may employ multiple monitors, with one monitor configured as a master monitor that can assign the workload across all the monitors. (Without losing generality, we assume one monitor per AS in the rest of the paper.)

Second, the monitor needs to be able to remotely login to each router that it is wired with and execute commands on the router, such as querying the AS path or the next hop from the router to any destination IP address. To do so, we assume every router supports secure shell (*ssh*), which is true for most routers nowadays, such as Cisco and Juniper routers [16], [17].

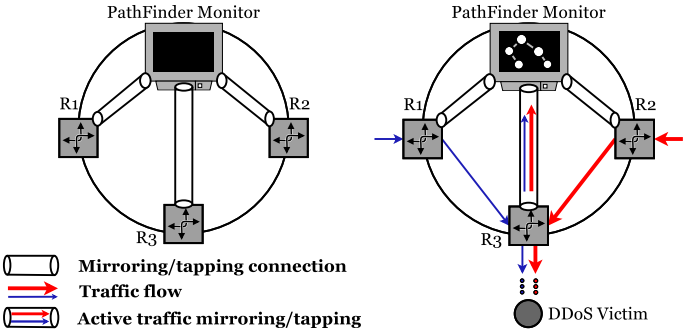
Finally, the monitor must be easy to discover by every PathFinder proxy. While the monitor can employ a running daemon process with a publicly known port number, proxies must also know the monitor's IP address. Many options exist; for example, the monitor can have its IP address and other information maintained at a web page. Or, the AS can set up a Domain Name System (DNS) record for its PathFinder monitor; e.g., `3582.pathfinder.org` may point to the PathFinder monitor of `AS 3582`.

C. Operation

The monitor at every PathFinder-participating AS operates on demand, remaining idle unless it receives a request from a PathFinder proxy, in which case the monitor will learn the IP address or prefix of the victim in question, together with the parameters in the request as defined in Sec. III-A, and start to generate PFLogs on behalf of the victim.

The very first step that the monitor takes is to identify a set of routers in its AS that are both sufficient and necessary to capture all the possible traffic that the AS may originate or forward toward the victim. Note the AS may also originate traffic toward the victim from any router of the AS itself. We therefore use all possible egress routers from which the traffic to the victim may exit the AS. For example, in Fig. 2b as the AS forwards two traffic flows toward the victim and both exit the AS from egress router $R3$, the monitor will select $R3$ to produce PFLogs for the victim. Further, it is straightforward to decide which routers are possible egress routers for the traffic to the victim. The monitor can query every border router's RIB to learn its next hop to reach the victim's IP. If the next hop is a router still within the AS, the border router in question is not an egress router; otherwise, it is an egress router.

Once the routers are selected, the monitor then talks with them to collect and produce PFLogs. First, the monitor will



(a) The setup of a PathFinder-participating AS.

(b) The snapshot when the monitor is capturing traffic.

Fig. 2: **An example setup of a PathFinder-participating AS.**

query each selected router to retrieve its AS path to the victim from its RIB. Furthermore, if the client requests the bandwidth consumption information of the total traffic to the victim via the AS, since the monitor is mirroring or tapping the traffic from these routers (among others), the monitor can observe the traffic from these routers and count their total volume (# of packets or # of bits), either per time unit or over a period of time (as specified in the request or based on a default value). Note that the mirrored or tapped traffic is only the traffic to the victim and more importantly, not on the path of the production traffic, therefore they will not impose a burden on the production traffic.

At this point, if the client did not request the source addresses or prefixes of the traffic, or source-based bandwidth consumption or other information, the monitor has collected all the PFLogs for the client, and thus can return them to the proxy of the client. We call this mode of operation **source-agnostic mode**. Otherwise, the monitor will operate in the **source-aware mode** to further produce source-based PFLogs via PFTrie (see Sec. V) before it returns them to the proxy.

Finally, note that each monitor only communicates with PathFinder proxies. No inter-AS collaboration is needed. In other words, monitors from different ASes are not required to communicate or collaborate, and each AS independently participates in PathFinder without any reliance on other ASes.

V. PFTrie—A PathFinder Data Structure for Traffic Logging

When in source-aware mode, the monitor at every PathFinder-participating AS will need to record all the sources that the AS has seen sending traffic to the victim in question, and if requested, the bandwidth consumption information per source. In doing so, the monitor needs to employ a data structure and accompanying algorithms to log sources, count bandwidth consumption, and transmit such data, all at a high speed to keep up with the line-speed packet arrival rate. Meanwhile, due to its speed, the trie data structure has been popular in storing IP addresses and prefixes, such as those in the Forwarding Information Base (FIB) of routers. A trie is also called a *prefix tree*, where every node on the trie uses its position on the tree to store the key of the node, such as the IP

address or prefix represented by the node. We therefore adopt the trie data structure for this purpose. Furthermore, to meet the design requirements discussed in Sec. IV-A, we enhance the trie data structure and design the PFTrie as follows.

A. Basic PFTrie Operations

The monitor captures and processes every packet toward the victim. It will make sure the IP source address of the packet is stored into the PFTrie via a ***put*** process. In this process, the monitor may modify the PFTrie by adding new nodes to store the IP, or discover that the address is already stored due to a previous packet with the same IP. The *put* process will return a node representing the IP, which the monitor can further update with bandwidth consumption information incurred by the current packet, if requested.

In the *put* process, the monitor will traverse the trie from the root downwards. It will traverse a node at each level—which we also call an *anchor*—to further move to the next level; clearly, when the traversal starts the anchor is the root of the trie. At the same time, it iterates through the bits of the IP address, starting from the leftmost bit, as follows:

(1) If the current bit in the IP address is `0`, it traverses to the left child of the anchor, otherwise it traverses to the right child; either way, the chosen child will become the new anchor.

(2) In case the new anchor does not exist, the monitor will detect a fault, i.e., the trie has not stored this IP address yet; the monitor will then add the missing child onto the trie, and use this child as the new anchor. (Note that this new anchor will also avoid the same fault for the next time.)

(3) If the current bit is already the rightmost bit of the IP address, the monitor then knows that the IP address is stored in the trie as represented by the new anchor, and thus return the new anchor node. Otherwise, it still needs to move to the next bit of the IP address; it then uses the new anchor as the current anchor to repeat step (1) above. Note the returned node is always a leaf node on the trie.

Fig. 3 shows an example of inserting a new IP address that ends with `101`. When it traverses to node a by following bit `1`, it needs to follow bit `0` to go to a's left child; since it does not exist, the monitor adds node b as a's left child. It then needs to follow the last bit `1` to go to b's right child; since it does not exist either, node c is then added, which also represents the newly stored IP address.

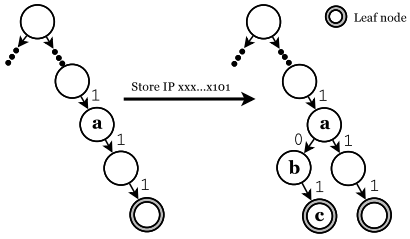


Fig. 3: **Store an IP address that ends with 101.**

B. Trie Optimization

We further optimize the PFTrie toward a faster traversal process. First, with the design in Sec. V-A, for every bit of an IP address, the trie must maintain a node at each level; for example, an IPv4 address will lead to 32 nodes at 32 respective levels on the PFTrie. We address this issue with two optimization methods that can go in parallel: the bottom-up aggregation of leaf nodes and the top-down collapse of prefixes. Furthermore, we also introduce a method to avoid duplicate traversal of the PFTrie when a source address is already stored. We describe each method below.

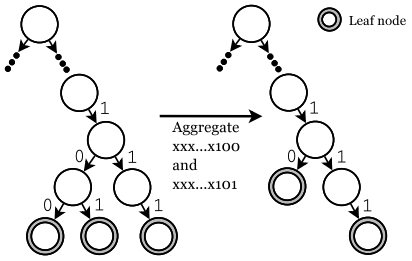


Fig. 4: **PFTrie optimization: Aggregating sibling leaf nodes into one new leaf node.**

1) Bottom-up Aggregation of Leaf Nodes: Because of traffic locality, sometimes there can be multiple sources from the same prefix sending traffic to the victim. For example, besides seeing the 32-bit source IP `xxx...x101` to the victim, the monitor may also see traffic to the victim from another source IP `xxx...x100`, which only differs from the former source IP by the very last bit; in other words, they share the same 31-bit prefix. When such locality is detected, two leaf nodes at level 32 are not needed to represent the two IP addresses. Instead, as shown in Fig. 4, we can aggregate the two leaf nodes into one new level-31 *leaf* node, indicating the monitor has seen traffic from both IP addresses in the 31-bit prefix. Furthermore, this aggregation can continue if the new leaf node has a sibling leaf node. Clearly, this bottom-up aggregation process can reduce the depth of certain branches of the PFTrie, thus speeding up the *put* process. One challenge here is the logging of the bandwidth consumption information. After aggregation, the monitor could simply copy the bandwidth consumption of each old leaf node into the new leaf node; or, it can also sum the bandwidth consumption of the two leaf nodes, thus recording the bandwidth consumption of the IP prefix represented by the new leaf node. The choice here depends on the client's request regarding the prefix granularity for recording the bandwidth consumption information (e.g., a /32 prefix granularity means to record the information per IP address, while a /0 prefix means the total bandwidth consumption for the whole IP space).

2) Top-down Collapse of Prefixes: We notice that in the *put* process the nodes at the top portion of the PFTrie are frequently traversed. Rather than traversing these nodes one by one each time, we collapse them into all the IP prefixes they represent, allowing the *sub-trie* below each prefix to be reached by directly indexing an array. Fig. 5 shows an example of collapsing /24 prefixes into an array (the monitor can collapse prefixes of other lengths, such as all the /16 prefixes, similarly). We populate the array with all /24 prefixes that exist in the PFTrie. For every /24 prefix, the monitor treats the 24

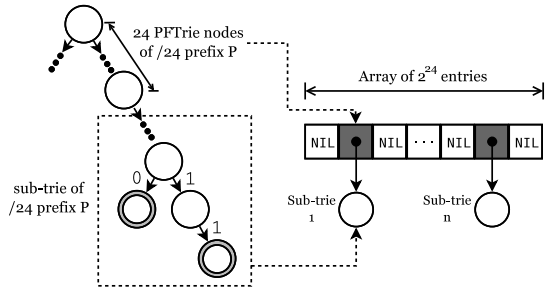


Fig. 5: **PFTrie optimization: Collapsing all /24 prefixes into an array with 2^{24} entries.**

bits of the prefix as an integer, use the integer as the index to directly locate the entry of the array, and have that entry point to the sub-trie originally below the prefix. So, instead of traversing 24 nodes of a /24 prefix and then traversing the sub-trie of the prefix, the monitor can immediately locate the entry for this prefix in the array, access from the entry the sub-trie of this prefix, and then traverse the sub-trie as before.

3) Avoidance of Duplicate Traversal: So far, if the PFTrie has recorded an IP address, when a packet with the same IP address arrives, the monitor will still run the *put* process and traverse the PFTrie, only to find the IP address is already stored. With n more packets from the same IP address, the extra overhead will be multiplied by n times.

We introduce a bitmap for each one of M most recently visited sub-tries. After storing an IP address in a sub-trie, the monitor will also set the bit in the bitmap corresponding to this IP to `1`, so that the *put* process for the same IP later will return very quickly. For example, the sub-trie for prefix `a.b.c/24` can have a bitmap of 2^8 bits, with the bit at index d corresponding to IP address `a.b.c.d`.

If we also need to update the bandwidth consumption information for the IP address (or its prefix), we will need to access its leaf node. To still have a speedy *put* process for duplicate IP addresses, we replace the bitmap with an array of pointers, and setting a bit to `1` above becomes inserting into the array a pointer that points to this leaf node. In the above example, the pointer at index d will be either *null* or point to the leaf node for IP address `a.b.c.d` (or its prefix).

VI. PathFinder Proxy

A. Addressing Design Requirements

The purpose of a PathFinder proxy is to learn the footprints of the traffic toward a victim. Since on the AS-path of the traffic there can be multiple ASes, when the monitors of such ASes report the AS-paths of the traffic, the AS-paths reported by them may overlap. The proxy must determine which AS-path includes the largest number of ASes. Furthermore, if source-based traffic footprint is requested, these monitors may also store and report the same IP address or prefix. The design of the proxy should resolve the potential conflict between monitors regarding the same IP address or prefix. Finally, the proxy's design should be cost-aware. Since the proxy does not contact monitors unless requested by a PathFinder client, clearly the best operation mode of the proxy is also on demand.

B. Setup

Every PathFinder proxy will make itself available to potential PathFinder clients. If a PathFinder client needs PathFinder service, it will register itself at a proxy, including setting up all security credentials. The client then can send a request to the proxy when it needs to obtain the traffic footprints of a DDoS victim.

We assume the proxy has a list of PathFinder-participating ASes (which the proxy can obtain, for example, through a web page). Further, it knows how to locate the PathFinder monitor of each AS, as described in Sec. IV-B.

C. Operation

Like any PathFinder monitor, every proxy also operates on demand. Once a proxy receives a request from a client, it will verify that the request is authentic and valid, and if so, learn who the victim is from the request and forward the request to monitors at PathFinder-participating ASes (or a subset of them if specified in the request).

If the footprints do not need to be source-based, each monitor will function in source-agnostic mode and the proxy will receive PFLogs from each monitor that contain the AS path from the monitor's AS to the victim, as well as bandwidth consumption information if requested. The proxy then adds the path to a path pool, with two exceptions: (1) If the path is just a part—i.e., a **sub-path**—of another path in the pool, the proxy can ignore this path. (2) Conversely, if a path from the pool is a sub-path of this path, the latter will replace the former in the pool. As a result, the proxy will learn a set of AS paths to the victim, and can return them to the client, together with the bandwidth information if requested.

However, if the footprints need to be source-based, the proxy will construct a local PFTrie based on the PFTries it receives from monitors. Every monitor incrementally transmits its PFTrie to the proxy; e.g., whenever it can fit newly added PFTrie nodes into an IP packet or a timer expires, the monitor will transmit the updates of its PFTrie to the proxy. For each leaf node of the PFTrie from each monitor, which represents an IP address or prefix `S` that the monitor has captured, the proxy will store `S` in its local PFTrie, following the same *put* process described in Sec. V-A. Furthermore, assuming the monitor's AS is `AS k`, the proxy also marks the leaf node that represents `S` with `k`, in order to indicate the AS-path of traffic from `S` to the victim is the AS-path from `AS k` to the victim. However, if `S` is already in the local PFTrie, the proxy will retrieve the marked AS number of the leaf node for `S`, say `o`, and mark the leaf node for `S` with either `k` or still `o`, whichever AS is upstream. We thus always can obtain the most complete AS-path from `S` to the victim.

VII. Evaluation

A. Goals

We now evaluate PathFinder in terms of the following:

- *Benefits of PathFinder for DDoS defense:* Since PathFinder footprints include path and traffic information, any DDoS

defense system that deploys filters inside the network to discard DDoS traffic can take advantage of the footprints to make better filter deployment decisions. We build a DDoS attack and defense simulation to study how PathFinder can benefit a DDoS defense system and make it more effective. We look at four different strategies of placing DDoS traffic filters and show that a DDoS defense system—when utilizing PathFinder—uses much less resource and achieves a much higher level of success.

- *Speed and Overhead of PathFinder:* At the core of PathFinder are the PFTrie-based operations at each monitor and proxy, particularly the *put* process. Therefore, we evaluate the time to store an IP address or prefix `S` in a PFTrie under two scenarios. In one scenario, `S` is new and the PFTrie needs to be updated with a set of new nodes, including the leaf node that represents `S`. In another scenario, `S` is in the PFTrie, so the *put* process will check and return the current leaf node that represents `S`. We call these two scenarios **put_a_new** and **put_an_old**, respectively. Furthermore, we evaluate the memory overhead of the PFTrie at each monitor and proxy when producing source-aware footprints and the network overhead when transmitting a PFTrie.

B. Benefits of PathFinder for DDoS Defense

To quantify the benefits of PathFinder for DDoS defense, we first define the model of DDoS attack and defense, and then compare the simulation results of a DDoS defense system with and without PathFinder.

1) DDoS Attack and Defense Model: The DDoS attack and defense model includes a botnet, an attack victim, a DDoS defense system, and the PathFinder system. The botnet contains 100,000 bots, where each bot is at a random tier-3 AS and has a fixed uplink bandwidth of 25 Mbit/s. The victim can at most handle 10 Gbit/s incoming traffic. During an attack, the victim applies the DDoS defense system to filter DDoS traffic, with or without the help of PathFinder to locate best locations for filters. We allow only tier 2 and 3 ASes that are close to the attack sources to deploy filters, as in practice the ASes close to the victim are subject to link congestion under DDoS attack (often too late for them to filter the DDoS traffic).

Since the same set of bots tend to take the same paths to send traffic toward the victim, these bots will no longer be effective once filters are deployed to filter their traffic. An intelligent attacker therefore would periodically switch to a new batch of bots to launch its attack. We define the *attack cycle* as the time window for every batch of bots used by the attacker. On the other hand, upon a newly seen DDoS traffic, we assume it takes T seconds in total for PathFinder to collect traffic footprints and also for the DDoS defense system to place the filters. In this study, we evaluate PathFinder system under the worst-case scenario where the attack cycle is no greater than T; in another words, before the DDoS defense places filters for the current bots attacking the victim, the attacker already switches to a new attack cycle with a new batch of bots.

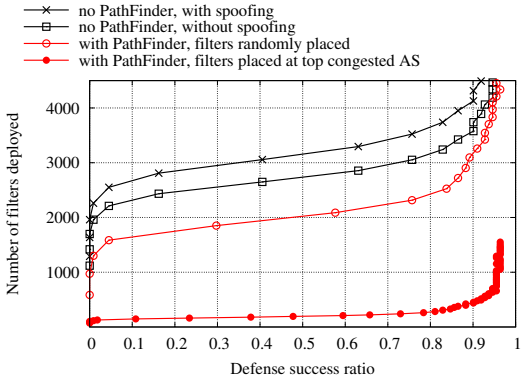


Fig. 6: **DDoS defense with and without PathFinder**

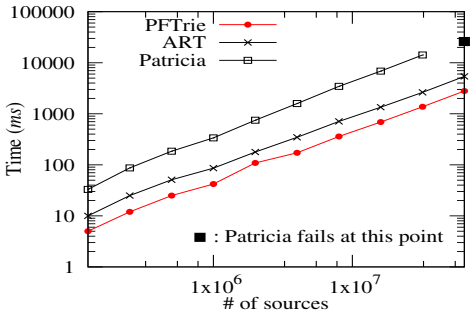
2) DDoS Defense Efficacy with PathFinder: We then compare the DDoS defense efficacy without and with PathFinder. When the DDoS defense does not have help from PathFinder, it uses the inferred AS-level topology to place filters, with two cases: 1) there is a 30% chance that a filter deployed is ineffective due to asymmetric routing on the Internet [10]; 2) further with some portion of the bots using IP spoofing, there is then a 50% chance that a filter will be ineffective. When PathFinder is in place, we use two AS selection methods for filter placement with the help of PathFinder: 1) randomly select an upstream AS; 2) select an AS that belongs to top k ASes that carry most of the DDoS traffic.

Figure 6 shows results for all four cases defined above. We see the results of DDoS defense system without PathFinder performs much worse than both cases when PathFinder is available. With PathFinder in place, and by applying filters at top congested ASes (which requires path and bandwidth information from PathFinder), the victim can survive 90% of the attack cycles with roughly 500 filters, whereas it takes at least 3,500 filters for a DDoS defense system to subdue 90% of the attack cycles if there was no PathFinder. In the case when the DDoS defense system uses only path information from PathFinder and places filters randomly at ASes, the system still uses a much smaller number of filters compared to the two defense cases without PathFinder.

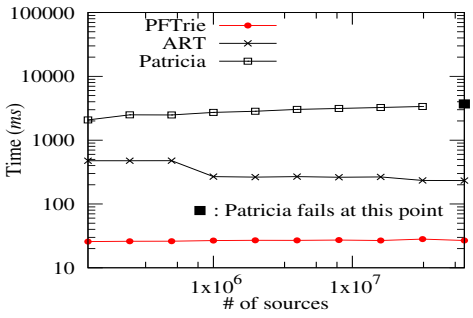
C. Speed and Overhead of PathFinder

1) Experiment Setup: To evaluate the PFTrie speed and its memory overhead, we used a desktop with Intel i7-4790 at 3.6 GHz with an 8-MB L3 cache and a 32-GB RAM at 1600 MHz. We implemented the PFTrie in *C*, and used the *Clang* compiler with the optimization level 2 to compile the code. We also created 50 synthetic traffic traces that contain 150 thousand to 64 million IP addresses; for each size we created five traces with different levels of source address locality, with 0%, 25%, 50%, 75%, and 100% addresses, respectively, that belong to the same IP prefix and can be aggregated.

2) Speed of PathFinder (i.e., PFTrie): We compare PFTrie's performance against Adaptive Radix Tree (ART) [18] and the well-known Generalized Prefix Tree [19] (also called Patricia Trie) under the two scenarios defined in Sec. VII-A. We use the synthetic traces that contain one to four magnitude



(a) put_a_new scenario.



(b) put_an_old scenario.

Fig. 7: **PFTrie speed.**

more IP addresses, in order to evaluate the three different data structures under stress.

Fig. 7a shows the comparison results under the put_a_new scenario for storing all IP addresses in a trace as new addresses into a data structure. For every synthetic trace size ranging from 160,000 to 64 million source addresses, when storing a new IP address or prefix, PFTrie always outperforms ART and Patricia. For example, to store 16 million IP addresses, it takes more than 1300ms for ART but it only takes around 700ms for PFTrie. In general, PFTrie spends 50% less time than ART to store the same number of IP addresses. Even to store 64 million IP addresses, it takes only 2.93s.

Fig. 7b shows results under the put_an_old scenario for performing 15 million *put* processes of storing an IP address or prefix already stored. Here, the time needed by PFTrie is virtually constant at about 27.0ms, much less than that in the put_a_new scenario; e.g., we can deduce with 64 million IP addresses, it would be about 115ms as opposed to 2.93s in the put_a_new scenario. Moreover, the time is also further less compared to ART and Patricia, and PFTrie is at least 10 times faster than ART in every case. This speed is because of the PFTrie optimizations we introduced, including the top-down collapse of prefixes (Sec. V-B2) and the avoidance of duplicate traversals (Sec. V-B3).

While the PFTrie operations are a combination of the two scenarios, from various real-world traces we notice that the put_an_old scenario is more frequent than the put_a_new scenario. For example, in the Booter 3 DDoS trace [20][21], the put_an_old scenario will happen 369 times more than the put_a_new scenario.

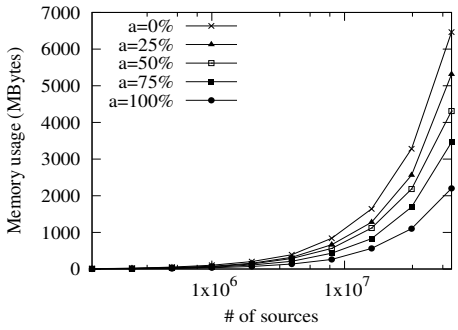


Fig. 8: **PFTrie memory overhead across five different source profiles.** Each profile has a different percentage of aggregatable addresses (a).

3) Overhead of PFTrie: We are particularly interested in the memory cost of PFTrie when there are millions of IP source addresses. We used synthetic traces that include a huge number of IP source addresses, and evaluated the memory usage for each number of sources under five different profiles, as shown in Fig. 8. We can see the logarithm of the memory cost is basically a linear function of the logarithm of the number of sources, and overall the memory cost is manageable under all five profiles. Moreover, a profile with a higher address locality can have much lower memory cost. This feature is due to the optimization via bottom-up aggregation of PFTrie leaf nodes (Sec. V-B1). Because of the tree nature of PFTrie, the memory cost complexity for storing 2^n addresses in a PFTrie is $O(2^n)$ with n levels of nodes, but if it shrinks to k levels, the memory cost will become $O(2^k)$, a reduction of $O(2^{n-k})$ times.

We also evaluated the network overhead across 25 different AS-level Internet topologies, using one million IP source addresses. For each AS-level topology, we assigned every IP address to an AS, where the number of addresses assigned to each AS is proportional to its IP address space size. Fig. 9 shows the network transmission overhead for an AS to transmit the PFTrie for 1 million source addresses to a proxy. Clearly, the further away an AS is from the victim, the smaller the network overhead it introduces. The AS that is the last hop to reach the victim would see traffic from all addresses, thus incurring the largest overhead, but only about 3.9MB.

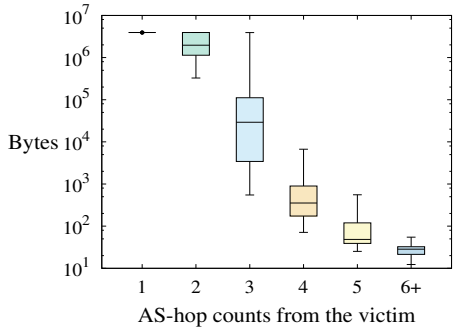


Fig. 9: **Network overhead in transmitting PFTries of 1 million source addresses.**

VIII. Discussions and Open Issues

PathFinder is an approach to obtaining the DDoS traffic footprints at the Internet scale, and we have made many design choices in order to provide a line-rate, cost-effective, and deployable solution. Nonetheless, some issues remain to be addressed due to space limitations:

One obvious issue is IP spoofing. Clearly, nothing would be affected if a monitor's mode of operation is source-agnostic, but if a PathFinder client requests source-based traffic footprints, the PathFinder system may learn some source IP addresses that are spoofed. An attacker may even generate many spoofed sources to overwhelm every monitor and the proxy of the client. We point out that even if a source address is spoofed, the path that the client learns about the source will still be valid, since the monitor that reported the source was on the path of the packet with the spoofed source. Furthermore, if the client notices multiple paths for the same source address or prefix, the client will know that either a routing change occurred, or at least some of them are spoofed sources.

We have also assumed that every AS (via its PathFinder monitor) is willing and able to communicate with any PathFinder proxy for the common good for DDoS defense. While we have shown the traffic overhead when a proxy communicates with every AS is not a concern (Sec. VII-C3), it is likely that this assumption is not true for *some* ASes due to incentive or connectivity issues, which we treat as an open issue out of the scope of this paper.

Another issue is to make PathFinder work for IPv6. In fact, the design of the PFTrie is independent of the length of an IP address and works for both IPv4 and IPv6. In the future, we plan to evaluate its speed and memory cost when handling millions of IPv6 addresses.

We do not fully discuss the security of PathFinder. We assume that every node in the PathFinder system must be authenticated before it can talk to other nodes in the system. We also assume that the system employs state-of-the-art defense mechanisms to protect itself against any security attacks; for example, a PathFinder proxy can employ a DDoS defense solution to protect itself and its communication with PathFinder monitors from DDoS attacks.

IX. Conclusions

While DDoS attacks have become more frequent and damaging and, once launched, can cause severe damage to services on the Internet, defense against DDoS attacks has often suffered from the lack of relevant knowledge of the DDoS traffic. However, it is fairly challenging to grasp the topological nature of the DDoS traffic while the attack is occurring: the DDoS traffic often originates from many different locations, follows various paths to reach the victim, sometimes carry spoofed source addresses, and can be extremely dynamic. Currently the best options are various IP traceback or path inference approaches, but they impose stringent demands to run and deploy. We fill this gap by proposing the PathFinder system as a service that a DDoS defense system can use to obtain the footprints of the DDoS traffic to a victim, including specifying many details of the footprints such as whether the source address and/or bandwidth information is needed. In particular, PathFinder embraces an architecture that not only eases its deployment in today's Internet, but also ensures it has a low cost (e.g., its on-demand model) and is fast to meet the line rate of the packets it must capture.

References

[1] Akamai. (2016) Q4 2016 state of the Internet security report. http://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp.

[2] K. York. (2016) Dyn statement on 10/21/2016 DDoS attack. http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack.

[3] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," in *USENIX Large Installation System Administration Conference (LISA)*, 2000, pp. 319–327.

[4] K. Singh, P. Singh, and K. Kumar, "A systematic review of IP traceback schemes for denial of service attacks," *Computers & Security*, vol. 56, pp. 111–139, 2016.

[5] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *ACM SIGCOMM*, 2000, pp. 295–306.

[6] A. Yaar, A. Perrig, and D. Song, "FIT: fast Internet traceback," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, 2005, pp. 1395–1406.

[7] K. J. Argyraki and D. R. Cheriton, "Active Internet traffic filtering: Real-time response to denial-of-service attacks," in *USENIX Annual Technical Conference*, 2005, pp. 135–148.

[8] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *ACM SIGCOMM*, 2001, pp. 3–14.

[9] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation," in *IEEE Symposium on Security and Privacy*, 2004, pp. 115–129.

[10] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On AS-level path inference," in *ACM SIGMETRICS*, 2005, pp. 339–349.

[11] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. E. Anderson, and A. Krishnamurthy, "Reverse traceroute," in *USENIX Symposium on Networked Systems Design and Implementation*, vol. 10, 2010, pp. 219–234.

[12] Cisco. (2016) Cisco IOS IP routing: BGP command reference. http://www.cisco.com/c/en/us/td/docs/ios/iproute_bgp/command/reference/irg_book/irg_bgp5.html.

[13] J. Stretch. (2016) JUNOS-to-Cisco IOS/XR command reference. http://web.archive.org/web/20140114070827/http://packetlife.net/wiki/junos-cisco-iosxr-command-reference.

[14] Cisco. (2016) Catalyst switched port analyzer (SPAN) configuration example. http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html.

[15] Juniper. (2016) Example: Configuring port mirroring for local monitoring of employee resource use on EX series switches. https://www.juniper.net/documentation/en_US/junos/topics/example/port-mirroring-local-ex-series.html.

[16] Cisco. (2007) Configuring secure shell on routers and switches running cisco ios. http://www.cisco.com/c/en/us/support/docs/security-vpn/secure-shell-ssh/4145-ssh.html.

[17] Juniper. (2015) Configuring SSH service for remote access to the router or switch. https://www.juniper.net/documentation/en_US/junos/topics/task/configuration/ssh-services-configuring.html.

[18] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: ARTful indexing for main-memory databases," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 38–49.

[19] D. R. Morrison, "PATRICIA—practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM*, vol. 15, no. 4, pp. 514–534, 1968.

[20] SimpleWeb.org. (2015) Traces. https://www.simpleweb.org/wiki/index.php/Traces.

[21] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - an analysis of ddos-as-a-service attacks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 243–251.

CellPAD: Detecting Performance Anomalies in Cellular Networks via Regression Analysis

Jun Wu[1], Patrick P. C. Lee[2], Qi Li[1], Lujia Pan[3,4], Jianfeng Zhang[4]
[1]Tsinghua University [2]The Chinese University of Hong Kong
[3]Xi'an Jiaotong University [4]Huawei Noah's Ark Lab

***Abstract*—How to accurately detect Key Performance Indicator (KPI) anomalies is a critical issue in cellular network management. We present CELLPAD, a unified performance anomaly detection framework for KPI time-series data. CELLPAD realizes simple statistical modeling and machine-learning-based regression for anomaly detection; in particular, it specifically takes into account seasonality and trend components as well as supports automated prediction model retraining based on prior detection results. We demonstrate how CELLPAD detects two types of anomalies of practical interest, namely sudden drops and correlation changes, based on a large-scale real-world KPI dataset collected from a metropolitan LTE network. We explore various prediction algorithms and feature selection strategies, and provide insights into how regression analysis can make automated and accurate KPI anomaly detection viable.**

***Index Terms*—anomaly detection, cellular network management, measurement and analysis**

I. INTRODUCTION

The continuing advances of cellular network technologies make high-speed mobile Internet access a norm. However, cellular networks are large and complex by nature, and hence production cellular networks often suffer from performance degradations or failures due to various reasons, such as background interference, power outages, malfunctions of network elements, and cable disconnections. It is thus critical for network administrators to detect and respond to performance anomalies of cellular networks in real time, so as to maintain network dependability and improve subscriber service quality. To pinpoint performance issues in cellular networks, a common practice adopted by network administrators is to monitor a diverse set of *Key Performance Indicators (KPIs)*, which provide time-series data measurements that quantify specific performance aspects of network elements and resource usage. The main task of network administrators is to identify any *KPI anomalies*, which refer to unexpected patterns that occur at a single time instant or over a prolonged time period.

Today's network diagnosis still mostly relies on domain experts to manually configure anomaly detection rules [25]; such a practice is error-prone, labor-intensive, and inflexible. Recent studies propose to use (supervised) *machine learning* for anomaly detection in cellular networks (e.g., [3], [8], [10], [11], [13], [34]) and search engines (e.g., [25]). However, machine-learning-based anomaly detection is subject to several well-known challenges [9], [25]: (i) the issues of which machine-learning algorithms should be used and how features should be configured depend on the actual anomaly detection problems and are difficult to address; (ii) labeling which time instants are anomalies for large-scale datasets is time-consuming; (iii) differentiating between normal data and anomalies is challenging and often requires domain knowledge to resolve; and (iv) anomalies occur much more infrequently than normal data, and this imbalanced nature can degrade learning accuracy [17].

ISBN 978-3-903176-08-9 ©2018 IFIP

In the context of cellular networks, we need to address additional challenges in anomaly detection. First, Internet traffic often exhibits periodic diurnal patterns [35] and different trends after long-term usage. In addition, the performance of cellular networks depends on not only the data transmission usage as in the traditional Internet, but also the radio resource usage [30]. Their corresponding KPIs, and hence anomalies, are often correlated. Such properties need to be properly addressed in the anomaly detection design. Thus, we are motivated to look into the following issues: (i) How should we define useful KPI anomalies that correspond to practical cellular network performance degradation problems? (ii) Can we design a unified anomaly detection framework that can incorporate various anomaly detection algorithms and detect various types of anomalies for one or multiple KPIs? (iii) Can our anomaly detection framework be automated with limited manual intervention, while still achieving accurate detection?

We present CELLPAD, a unified performance anomaly detection framework for cellular networks. CELLPAD builds on *regression analysis*, which predicts the expected quantities of KPI time-series data so as to provide prediction results for anomaly detection. We consider two types of anomalies that are of practical interest to cellular network management based on our internal communication with network administrators: *sudden drops*, which indicate the unexpected degradations of a KPI, and *correlation changes*, which indicate the inconsistency between the current and historical correlations of two correlated KPIs. Using CELLPAD, we conduct trace-driven evaluation to demonstrate how regression analysis achieves automated and accurate KPI anomaly detection. To summarize, this paper makes the following contributions:

- We first present a trace-driven analysis on a large-scale KPI dataset from a real-world metropolitan LTE network. Our dataset spans six KPIs, 17 weeks of duration, and 12,463 cells. We show the presence of anomalies in the dataset and motivate the practical need of anomaly detection.
- We design CELLPAD for anomaly detection in cellular net-

works. CELLPAD supports various prediction algorithms, including simple statistical modeling, linear regression, and tree-based regression (the latter two belong to machine-learning-based regression). In particular, it takes into account both seasonality and trend components in KPI time-series data, and provides a feedback loop for retraining the prediction models using prior detection results to improve detection accuracy.

- We conduct trace-driven evaluation on CELLPAD based on our KPI dataset to explore a range of prediction algorithms and different feature selection strategies. We also show that CELLPAD achieves more accurate sudden drop detection than Twitter's time-series anomaly detection tool [2]. We make several observations, such as the accuracies of different prediction algorithms, the robustness against parameter choices, and the importance of prediction model retraining for accurate anomaly detection. We find that no single prediction algorithm is an absolute winner in both sudden drop and correlation change detection.

The source code of CELLPAD is available for download at **http://adslab.cse.cuhk.edu.hk/software/cellpad**.

The rest of the paper proceeds as follows. Section II presents the background details and analysis of our KPI dataset and motivates the need of anomaly detection. Section III presents our design of CELLPAD. Section IV evaluates different prediction algorithms and design choices of CELLPAD. Section V reviews related work. Finally, Section VI concludes the paper.

II. DATASET

In this section, we provide an overview of the KPI dataset that we collected from a production cellular network. We also motivate the need of detecting anomalies in such a network.

A. LTE Network Architecture

In this work, we focus on the 4G LTE cellular technologies. We first provide a high-level overview of an LTE network architecture. An LTE network comprises three main entities: User Equipments (UEs), the Radio Access Network (RAN), and the Evolved Packet Core (EPC). Each UE refers to a user's mobile device. The RAN comprises multiple base stations called Evolved NodeBs (eNodeBs), each of which manages the radio resources of UEs and provides UEs with wireless connectivity. The EPC comprises the Mobility Management Entity (MME), the Serving Gateway (SGW), and the Packet Data Network Gateway (PGW): the MME manages UEs' control-plane functions (e.g., user authentication, mobility management), while both the SGW and PGW manage UEs' data-plane functions (e.g., data routing). To send or receive data via the Internet, a UE first sets up a radio connection with an eNodeB and a signaling channel with the MME. It then sets up a data session with the EPC atop the radio connection, and uses the data session for data transmission.

Each eNodeB serves multiple geographical areas called *cells*, each of which covers a number of UEs. The size of each cell depends on the local user population and the radio coverage. A production LTE network typically covers thousands of cells.

TABLE I
DESCRIPTIONS OF SIX CELL-LEVEL KPIS.

KPIs	Descriptions
USER	It refers to the number of active users.
RRC	It refers to the number of radio resource control (RRC) connection requests between a UE and an eNodeB. Each RRC connection works at the control plane and carries signaling messages for managing the radio resources of the UE.
ERAB	It refers to the number of E-UTRAN Radio Access Bearer (ERAB) requests between a UE and the EPC. Each ERAB works at the data plane and carries the data traffic of the UE.
PRB	It refers to the number of physical resource blocks allocated. It indicates the radio resource usage.
THR	It refers to the data transmission throughput in the downlink direction.
DUR	It refers to the duration of active data transmission in the downlink direction.

B. Data Collection

Network administrators deploy probes in the EPC and every eNodeB to periodically collect KPI values, which will be sent to a centralized network management system (NMS). We call each collected input an *instance*, which specifies the time and value for a KPI. In this work, we collected per-cell KPI instances from the NMS of an operational LTE network deployed in a metropolitan city in China. Each instance is recorded on an hourly basis and describes the performance of a cell in the latest hour. We consider six types of KPIs, as summarized in Table I. The six types of KPIs address the cellular network performance in three aspects: (i) user population (i.e., USER), (ii) radio resource usage (i.e., RRC, ERAB, and PRB), and (iii) data transmission load (i.e., THR and DUR).

Our KPI dataset covers three collection periods for a total of 17 weeks: (i) November 7, 2016 to January 8, 2017, (ii) February 13, 2017 to March 12, 2017, and (iii) April 10, 2017 to May 7, 2017. We only select the cells that have the complete KPI data over the entire 17 weeks; in other words, each cell has a total of $24\times7\times17=$ 2,856 instances for each of the six KPIs. Finally, we identify 12,463 cells. To the best of our knowledge, our dataset is among the largest being studied in the literature (in terms of the collection period and the number of cells being covered) regarding KPI measurements in operational LTE networks.

C. A First Look at the Dataset

We first examine the statistical properties of our collected dataset, so as to understand the behaviors of the cellular network. Our observations are summarized as follows: (i) there exist strong seasonality and trend components in the dataset; (ii) some KPIs are strongly correlated; and (iii) there exist non-negligible variances in KPI values across the same hour of different days.

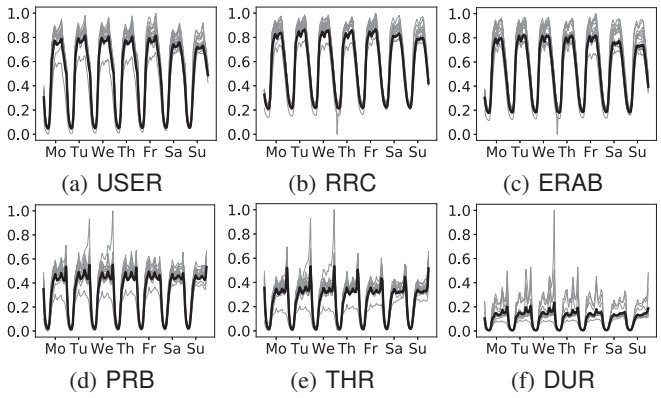


Fig. 1. Seasonality components of six KPIs. The x-axis represents the 168 hours of a week, and the y-axis represents the weekly normalized aggregated KPI value in each hour. We plot each week of KPIs separately in grey, while the black curve represents the average of 17 points in each hour.

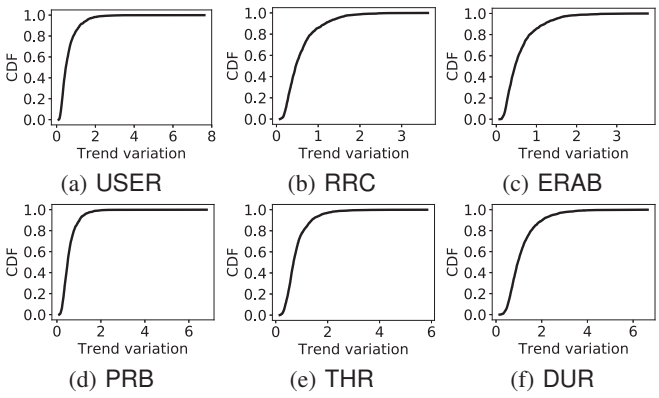


Fig. 2. Trend components of six KPIs. The x-axis represents the trend variation, and the y-axis represents the cumulative density function.

Seasonality: We first analyze the seasonality component (i.e., the recurring patterns over a time series) in all six KPIs. We first aggregate the KPI values at each hour across all cells. We then normalize each aggregate result x to the range $[0, 1]$ as $\frac{x-\min\{x\}}{\max\{x\}-\min\{x\}}$, where $\max\{x\}$ and $\min\{x\}$ represent the maximum and minimum of all 2,856 hours, respectively. Figure 1 plots the weekly normalized aggregate results for all 17 weeks. We see that all six KPIs show fairly stable diurnal patterns, albeit some abrupt increases or drops in some hours.

Trend: We next study the trend component (i.e., the increasing or decreasing patterns over a time series) in all six KPIs. We compute the trend component on a per-cell basis. Specifically, for each KPI, we compute the average KPI value of a cell at the i-th hour, denoted by y_i, over a sliding time window of 168 hours (a week) using the recent past and future KPI values, starting from the (i-84)-th hour to the (i+83)-th hour, where $i \geq 84$. We then compute the *trend variation* as $\frac{\max\{y_i\}-\min\{y_i\}}{\bar{y_i}}$, where $\max\{y_i\}$, $\min\{y_i\}$, and $\bar{y_i}$ denote the maximum, minimum, and mean of the sequence of y_i's, respectively. In our analysis, we pick the first time period from November 7, 2016 to January 8, 2017, in which we can compute 1,344 y_i's over the 9-week period. Intuitively, if the trend variation is close to zero, then the time series remains stable across any weekly cycle; otherwise, the time series has a strong trend component. For example, if the trend variation is larger than one, it means that the maximum differences between the average KPI values of any sliding windows can be larger than the overall average KPI value. Figure 2 shows the cumulative distribution of the trend variations of all cells for each KPI. We see that for each KPI, the trend variation is larger than one for a non-negligible fraction of cells.

Correlation: KPIs may be correlated; for example, if the number of active users increases, both the radio resource usage (i.e., RRC, ERAB, and PRB) and the data transmission load (i.e., THR and DUR) also increase. We compute the Pearson coefficient (PC) (a measure of linear correlation of two variables) for every pair of KPI time-series data of each cell, and obtain the average PC across all cells. If the PC is closer to 1.0, it implies that the two KPIs have high positive linear correlation. Table II shows the results. We observe all six KPIs have positive linear correlation. In particular, the pairs (RRC, ERAB), (PRB, THR), and (USER, RRC) are the top-3 pairs with the strongest correlation.

TABLE II
AVERAGE PEARSON COEFFICIENTS OF KPI PAIRS ACROSS ALL CELLS.

	USER	RRC	ERAB	PRB	THR	DUR
USER	1.000	0.895	0.907	0.829	0.771	0.817
RRC	-	1.000	0.961	0.709	0.602	0.654
ERAB	-	-	1.000	0.716	0.610	0.659
PRB	-	-	-	1.000	0.942	0.814
THR	-	-	-	-	1.000	0.776
DUR	-	-	-	-	-	1.000

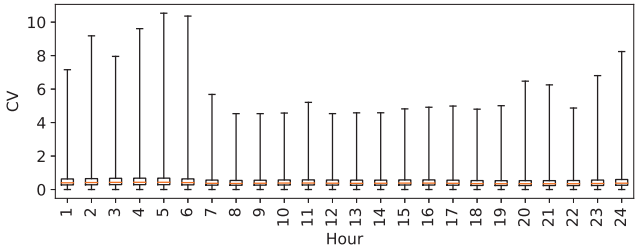
KPI variations: KPI values may fluctuate over time due to performance changes in cellular networks, thereby implying the presence of performance anomalies. To understand the frequency of such KPI variations, we calculate the coefficient of variation (CV) (i.e., the ratio of the standard deviation to the mean) of a KPI at each hour of a day for each cell. A large CV implies that the specific cell has a high deviation of the KPI. Here, we focus on USER. Figure 3(a) shows the boxplots[1] of CVs across all cells. We observe that the majority of CVs are close to zero, yet a few cells exhibit high CVs. Interestingly, we observe higher CVs during nighttime (from 23:00 to 06:00) than during daytime (from 08:00 to 18:00).

We also observe significant KPI variations in the correlations across a KPI pair in some of the cells. We calculate the PC of a KPI pair at each hour of a day for each cell. We focus on USER and RRC (which show a high PC according to Table II). Figure 3(b) shows the boxplots of PCs across all cells. While the majority of cells show a high PC (close to one), some cells show a negative PC, which is unexpected and may be anomalies.

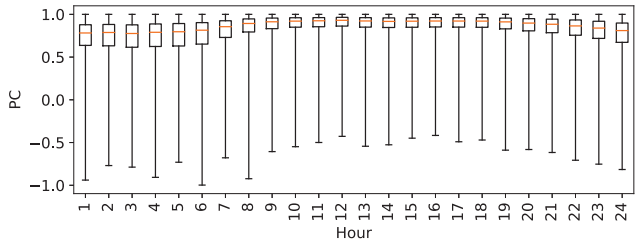
D. Definitions of Anomalies

Based on our analysis and internal communication with network administrators, we study two types of KPI anomalies,

[1]A boxplot shows the minimum, first quartile, median, third quartile, and maximum of all samples.



(a) Coefficients of variation (CVs) of USER



(b) Pearson coefficients (PCs) of USER and RRC

Fig. 3. KPI variations, in terms of boxplots at different hours of a day across all cells. Here, the x-axis represents the hour of a day (e.g., 1 means 0100).

namely *sudden drops* and *correlation changes*, that are of practical interest to cellular network management. A sudden drop refers to the sudden performance degradation of a KPI instance within a cell. For example, if there exists a sudden drop in USER, it may imply that a cell fails to provide connectivity to a significant portion of users. In general, a sudden drop happens when a KPI value is significantly less than the expected one. On the other hand, a correlation change refers to the large deviation of two correlated KPI instances within a cell. For example, a cell failure may increase the number of RRC request attempts (i.e., RRC), while the number of active users (i.e., USER) remains relatively unchanged. Thus, both sudden drops and correlation changes are complementary to each other in characterizing performance anomalies of cellular networks. In practice, if either one of the KPI anomalies persists for a prolonged period (e.g., a few hours), it may indicate the presence of network failures and requires network administrators to investigate further. In the following discussion, we propose a unified framework that can effectively detect both sudden drops and correlation changes.

Our anomaly detection focuses on a per-cell basis by inspecting the time-series instances of multiple KPIs in each cell. In this work, we do not consider the correlation across multiple cells. Also, we do not identify the root causes of the anomalies due to insufficient information in our dataset. We pose these issues as future work.

III. Design

We present CellPAD, a cellular network performance anomaly detection framework. It takes the time-series data of multiple KPIs as inputs, and detects both sudden drops and correlation changes with high accuracy by taking into account both seasonality and trend components in KPI time-series data. It also provides a feedback loop to incrementally update the prediction models based on the past detection outputs, thereby eliminating the manual efforts of specifying labeled data (i.e., ground truths) for model training.

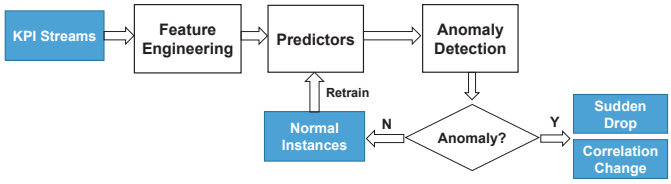


Fig. 4. CellPAD architecture.

A. Main Idea

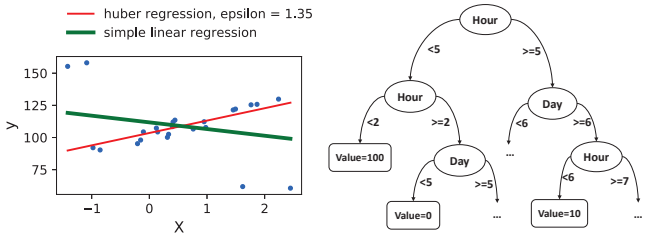
CellPAD builds on regression analysis to predict the normal values of KPI instances in order to detect anomalies. Figure 4 shows the CellPAD architecture, which provides a unified regression framework for detecting both sudden drops and correlation changes. At a high level, CellPAD takes multiple time-series streams of KPI instances at different time intervals (hours in our case) as inputs. It first performs feature engineering to extract a set of *features*, whose values are derived from the KPI instances that are observed up to the current hour. The feature values serve as inputs to different *predictors*, each of which performs a specific prediction algorithm and outputs a predicted KPI value, which is the expected value for a KPI at each hour in normal situations (i.e., without anomalies). For sudden drop detection, CellPAD returns one predicted KPI value for each KPI instance being considered, while for correlation change detection, it returns two predicted KPI values for each pair of KPI instances being considered. Finally, CellPAD performs anomaly detection based on the prediction at each hour by checking the deviations between the actual and predicted KPI values. It concludes that the current KPI instances are either anomalies (i.e., sudden drops or correlation changes) or normal instances. For the latter case, CellPAD also feeds back the normal instances to retrain the prediction models for improved detection accuracy.

One major design issue is to properly select the predictors and features. In particular, the features depend on not only what types of anomalies (sudden drops or correlation changes) being detected, but also the predictors being used. In the following, we formulate the regression framework of CellPAD in detail, in which we first state the predictors that CellPAD supports, followed by the corresponding feature engineering procedures.

B. Predictors

CellPAD supports three families of predictors: simple statistical modeling, linear regression, and tree-based regression; the latter two belong to machine-learning-based regression approaches. Each predictor returns a predicted value for each hour based on the underlying prediction algorithm. Here, we summarize the algorithms that we consider under each family.

Simple statistical modeling: CellPAD implements four algorithms:



(a) Huber regression (b) Regression tree

Fig. 5. Regression applied in CELLPAD

- *EWMA (Exponentially Weighted Moving Average)* [19]: It computes the predicted value based on the weighted values of a set of instances, such that the weights are exponentially decayed for older instances.
- *WMA (Weighted Moving Average)* [29]: Its prediction is also based on the weighted instances as in EWMA, except that the weights are linearly decayed.
- *HW (Holt-Winters)* [37]: It is a triple exponential smoothing method that extends EWMA to deal with seasonality and trend. It computes the predicted value as a function of the weighted inputs of both instances as well as the seasonality and trend components. It also estimates the seasonality and trend components from the instances using EWMA.
- *LCS (Local correlation score)* [28]: It measures the correlation of two time-series. It holds two synchronous sliding windows to compute the auto-covariance matrices continuously and then aggregates the matrices using their exponentially weighted averages. We mainly use LCS for detecting correlation changes.

Linear regression: CELLPAD implements two linear regression algorithms to model the linear relationships between features and predicted values:

- *SLR (Simple linear regression)* [5]: It computes the predicted values based on the optimal linear combination of the values of a feature that can minimize the mean square deviation.
- *HR (Huber regression)* [21]: It enhances simple linear regression to be robust against noise, by controlling whether instances are classified as outliers via an epsilon parameter (a smaller epsilon is more robust to outliers). For example, Figure 5(a) shows how Huber regression excludes outliers from modeling as opposed to simple linear regression.

Tree-based regression: To model the non-linear relationships between features and predicted values, CELLPAD also implements two tree-based regression algorithms:

- *RT (Regression tree)* [7]: It organizes the feature space into a tree structure, in which each non-leaf node is a decision-making process that splits the feature space based on a selected feature, while each leaf node holds a local predictor that averages all instances that fall into the feature partition. Figure 5(b) shows a regression tree example, in which we choose the hour and day indexes as the features (see Section III-C for details). The predicted value is 10 if the features satisfy "(Hour $==$ 5) and ($6 \leq$ Day ≤ 7)". Choosing which feature for decision making and how to split the feature space can be controlled by a set of parameters, which we omit details here.
- *RF (Random forest)* [6]: It is an ensemble learning algorithm. It samples different subsets of instances and features to form multiple regression trees and take their average prediction result. It is robust against irrelevant features and noises than a single regression tree in general.

Discussion: Simple statistical modeling is easy to implement, as it can return the predicted values based on the observed instances. However, it has the major limitation that the prediction accuracy heavily depends on the parameter settings. In contrast, both linear regression and tree-based regression are less dependent on parameters, which can be "learned" from input instances. However, they require careful feature engineering for the regression analysis, as we will explain in the next subsection.

C. Feature Engineering

We now elaborate the feature engineering process for linear regression and tree-based regression. CELLPAD extracts different features for sudden drop and correlation change detection. We also describe how we address seasonality and trend.

Sudden drops: CELLPAD uses two types of features for sudden drop detection. The first type is called *indexical features*, in which we use the time indexes of each KPI instance as features. To take into account seasonality, we index the hour and day from 0 to 23 and from 0 to 6, respectively, and use the hour and day indexes as the features (called Hour and Day, respectively). Intuitively, if we group the instances by the same Hour only, we capture daily seasonality; if we group the instances by both the same Hour and Day, we capture weekly seasonality. In this work, we mainly focus on weekly seasonality. The indexical features are mainly used by tree-based regression (see Figure 5(b)).

The second type is called *numerical features*, in which we apply some numerical operations to KPI instances to extract features. We define each numerical feature as $\langle win, oper \rangle$, in which we run $oper$ on the KPI instances in the past win weeks. For instance, $\langle 5, mean \rangle$ means that we take the mean of KPI values in the past five weeks. By sampling different values of win and types of $oper$, we can generate a number of numerical features. To account for weekly seasonality, we only pick the KPI instances with the same Hour and Day. The numerical features can be used by both linear regression and tree-based regression.

Correlation changes: For correlation change detection (say, for KPI1 and KPI2), CELLPAD trains two predictors, one for KPI1 and one for KPI2. The predictor for KPI1 (resp. KPI2) takes the value of the current instance of KPI2 (resp. KPI1) as a feature. The rationale is that if the two KPIs are correlated, each KPI instance is dependent on another KPI instance at any given time. Linear regression uses this feature for prediction, while tree-based regression additionally takes Hour and Day as features for prediction.

Trend removal: As the changes in the KPI values caused by the trend component affect anomaly detection accuracy, we provide an option of removing the trend component from the raw KPI time-series. CELLPAD removes the trend component before extracting the features based on the idea of time-series decomposition [22]. Specifically, for a given KPI instance at some hour, CELLPAD computes the average KPI value of over a sliding window of 168 hours using the recent past and future KPI values as in Section II-C (note that we do not start anomaly detection until we collect enough past KPI instances for trend removal). To remove the trend component, CELLPAD divides the raw KPI value by the computed average value and feeds the result to feature engineering. Note that we can treat the trend component as additive or multiplicative, yet we choose the latter as it achieves better detection accuracy after trend removal based on our experience. We study the effect of trend removal in Section IV.

D. Anomaly Detection

To perform anomaly detection, we first calculate the degree of deviation. For sudden drop detection, CELLPAD computes the *drop ratio* $D = \frac{\text{KPI}_a - \text{KPI}_p}{\text{KPI}_p}$, where KPI_a and KPI_p denote the actual and predicted KPI values, respectively. If D is much less than 0, it likely implies a sudden drop. To detect correlation changes of two KPIs (say, KPI1 and KPI2), CELLPAD computes the *change ratio* for KPI1 by $C_1 = \frac{\text{KPI1}_a - \text{KPI1}_p}{\text{KPI1}_p}$, and that for KPI2 by $C_2 = \frac{\text{KPI2}_a - \text{KPI2}_p}{\text{KPI2}_p}$, where KPI1_a and KPI1_p (resp. KPI2_a and KPI2_p) denote the actual and predicted KPI values of KPI1 (resp. KPI2), respectively.

CELLPAD uses the "N-sigma rule" for anomaly detection, in which an anomaly is expected to deviate from the mean by a significant number N of standard deviations. At each hour, we calculate the mean μ and standard deviation σ for the drop ratios or change ratios in the last 168 hours. We call a KPI instance a sudden drop if $D < \mu - N\sigma$, and call two KPI instances a correlation change if $C_1 \notin [\mu - N\sigma, \mu + N\sigma]$ or $C_2 \notin [\mu - N\sigma, \mu + N\sigma]$. By default, we set $N = 3$, yet we also consider different values of N for the threshold selection.

Finally, CELLPAD outputs the anomalies, or feeds back the remaining normal instances to retrain the prediction model (see Figure 4), which extracts features from the normal instances for prediction.

IV. EVALUATION

We have implemented a CELLPAD prototype in Python. For EWMA, WMA, and LCS, we implement their algorithms directly; for HW, we use the open-source code [26], which selects the optimized weights that minimize a loss function; for SLR, HR, RT, and RF, we implement them using scikit-learn [1].

We evaluate the anomaly detection accuracy of CELLPAD, and compare CELLPAD with Twitter's open-source time-series anomaly detector [2] (called TWITTER for short). We address the following questions: (i) What is the accuracy of different predictors in sudden drop and correlation change detection? (ii) How do seasonality and trend affect detection accuracy? (iii) How is CELLPAD compared with TWITTER?

A. Methodology

It is a labor-intensive task for network administrators to identify real anomalies (i.e., labels) from our dataset, which is large and complex by nature; the same problem is also reported by previous work [3], [8], [24], [36]. Thus, we resort to injecting synthetic anomalies into the raw data of our dataset for evaluation. Specifically, we randomly select 80 cells from our dataset for evaluation. We aggregate the three collection periods into a continuous 17-week period (see Section II-B). In each cell, we randomly pick 1.5% of hours and three continuous segments with a uniformly distributed length of 3 to 24 hours each to inject anomalies. For sudden drops, we decrement the KPI values of each anomaly hour by a percentage uniformly distributed from 30% to 100%. For correlation changes, we pick one of the two KPIs of each anomaly hour, and either increments or decrements its value by a percentage uniformly distributed from 30% to 100%.

We also apply a simple rule-based method to label the obvious anomalies from the dataset based on the raw values. For sudden drops, we treat a KPI instance whose raw value is 75% smaller than either one of the KPI values at the same hour and day in the past two weeks as a sudden drop. For correlation changes, we compute and rank the ratios of the values of all KPI instance pairs, and treat the top 0.5% and lowest 0.5% of pairs as correlation changes. Finally, we have roughly 3-4% of anomalies in the whole 17-week dataset in each cell, and this percentage is consistent with the real-world scenarios based on our internal discussion with network administrators.

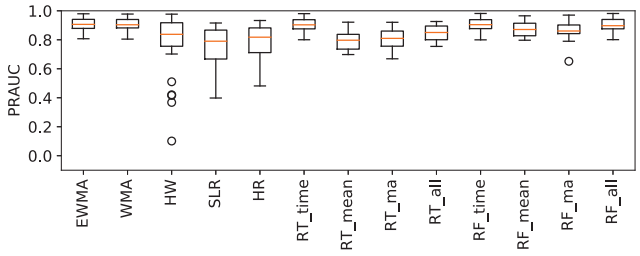
We use the first two weeks of KPI instances, including both normal instances and synthetic anomalies, to bootstrap our predictors. We then start our evaluation from the third week onwards. We do not exclude the synthetic anomalies in our bootstrapping process; instead, we rely on prediction model retraining to improve the robustness of our prediction.

B. Sudden Drop Detection

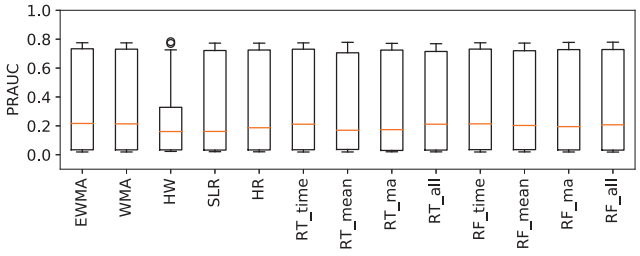
We first evaluate CELLPAD in sudden drop detection. We consider the metric *PRAUC (Area Under Precision-Recall Curve)*, which is shown to be robust when the distributions of normal instances and anomalies are highly imbalanced [15]. Here, we use the drop ratio (see Section III-D) as the prediction input to PRAUC, which computes various precision and recall pairs against different thresholds to obtain an accuracy measure between 0 and 1 (higher means more accurate). We only present the results for the KPI USER.

We consider the following predictors:

- *EWMA, WMA, and HW*: We compute the average using the values with the same hour and day indexes from the first week to the previous week. For EWMA, we set the weight to 0.8; for WMA, the weights are set based on the number of previous weeks; for HW, we set the seasonal period as 168 weeks and use it to compute the optimized weights [26].



(a) With trend removal



(b) Without trend removal

Fig. 6. PRAUC of different predictors in sudden drop detection.

- *SLR and HR*: The features are the mean and median of the values with the same hour and day indexes in the past w weeks, where w is sampled at $w = 3, 5, 7, 10, 13$; for HR, we set $\epsilon = 1.35$.
- *RT and RF*: We consider four variants for each of RT and RF. (i) RT_time and RF_time, which use the hour and day indexes as indexical features; (ii) RT_mean and RF_mean, which use the mean and median features as in SLR; (iii) RT_ma and RF_ma, which use the moving averages of both EMWA and WMA as features; and (iv) RT_all and RF_all, which use all features as described in (i), (ii), and (iii). For RF, we set the number of trees as 100.

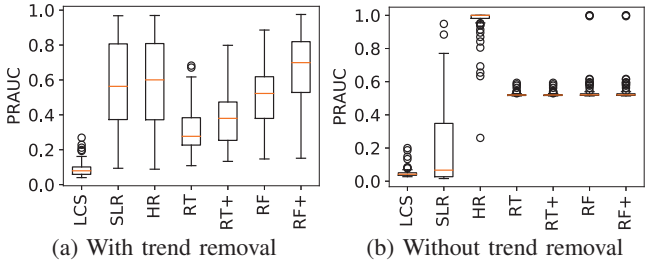
Figure 6 shows the boxplots of PRAUC for different predictors. Figure 6(a) first considers the case in which we remove the trend components. Simple statistical modeling and tree-based regression generally achieve good accuracy; for example, EWMA, WMA, RT_time, RF_time, and RF_all have an average PRAUC of more than 0.9. On the other hand, HW, SLR, and HR have low accuracy, with an average PRAUC of below 0.8. We note that RF maintains high accuracy using different features (with a mean of at least 0.86).

Figure 6(b) shows the results when we do not remove trend components. We see that the accuracy of all predictors drops significantly. This justifies the necessity of removing trend components in sudden drop detection.

C. Correlation Change Detection

We now study correlation change detection, in which we consider the following predictor implementations in CellPAD:

- *LCS*: We set the sliding window size as 20 hours and the smoothing constant as 0.8.
- *SLR and HR*: For each of the predictors of a KPI, we set the value of another KPI as the only feature.



(a) With trend removal (b) Without trend removal

Fig. 7. PRAUC of different predictors in correlation change detection.

- *RT and RF*: We consider two variants for each of RT and RF. (i) RT and RF, which use the value of another KPI as the only feature as in SLR and HR; and (ii) RT+ and RF+, which use the value of another KPI as a feature as well as the hour and day indexes as the indexical features. The rationale of using indexical features in RT+ and RF+ is to take into account weekly seasonality.

We use PRAUC as the accuracy metric. We use the average of two absolute change ratios $\frac{1}{2}(|C_1|+|C_2|)$ (see Section III-D) as the input to PRAUC. Here, we focus on the KPI pairs (USER, RRC).

Figure 7 shows the boxplots of PRAUC for different predictors. Depending on the predictors, the accuracy may be improved or degraded with trend removal. As opposed to sudden drop detection, RF does not achieve high accuracy here, even though using different features. Overall, HR without trend removal (i.e., using the raw KPI data for anomaly detection) achieves the highest PRAUC (with a mean 0.93).

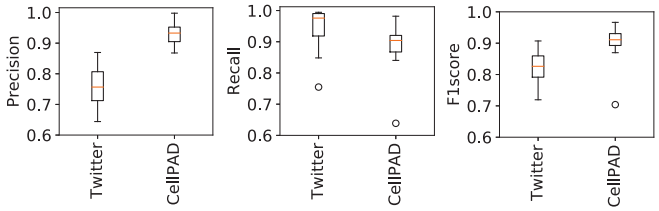
D. Comparisons with Twitter

We now compare CellPAD with Twitter [2] in sudden drop detection. Twitter is an open-source anomaly detection system that also takes into account the seasonality and trend components in the anomaly detection of time-series data. Since Twitter is designed for anomaly detection in a single time-series (as opposed to two time-series in correlation change detection), we only focus on sudden drop detection. Also, Twitter only tells if a time point is an anomaly, but does not return an anomaly measure for us to compute PRAUC for different thresholds. Thus, we consider the following accuracy metrics instead: (i) precision, (ii) recall, and (iii) F1-score (i.e., 2×Precision×Recall / (Precision + Recall)). For CellPAD, we pick RF_all (with trend removal) as the predictor.

Figure 8 compares CellPAD and Twitter in sudden drop detection for the KPI USER. CellPAD has much higher precision than Twitter, but with slightly lower recall. Overall, CellPAD achieves higher F1-score than Twitter (with means 0.90 and 0.82, respectively). One possible reason is that Twitter builds on statistical modeling, while CellPAD uses random forest regression here to achieve high accuracy; we pose further investigations as future work.

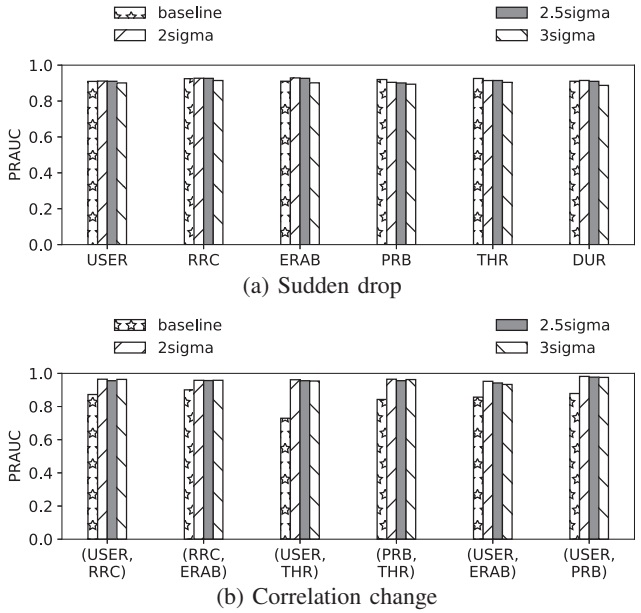
E. Effects of Model Retraining

Finally, we study the effect of retraining the predictor by feeding back the prior detection results. Here, we consider



(a) Precision (b) Recall (c) F1-score

Fig. 8. Comparisons between CELLPAD and TWITTER.



(a) Sudden drop

(b) Correlation change

Fig. 9. Effects of model retraining.

two cases: (i) the baseline case, which uses all instances (including normal instances and anomalies) to update the predictor and (ii) our CELLPAD design, which uses only the normal instances to update the predictor. In addition, we test different thresholds in anomaly detection by varying the number of standard deviations from the mean; here, we consider 2σ, 2.5σ, and 3σ. We use RF_all (with trend removal) and HR (without trend removal) as the predictors for sudden drop detection and correlation change detection, respectively.

Figure 9 shows the results for both sudden drop and correlation change detection; the former considers all six KPIs, while the latter considers six KPI pairs which show high PC (see Table II). We make the following observations. First, both RF_all and HR maintain high accuracy for different KPIs and KPI pairs in sudden drop detection and correlation change detection, respectively. Second, the baseline and CELLPAD do not show significant difference in sudden drop detection, while CELLPAD achieves higher accuracy than the baseline in correlation change detection. This justifies the need of retraining the predictor using normal instances only. Finally, we do not see significant difference for different thresholds in CELLPAD, meaning that CELLPAD remains robust in threshold selection.

F. Summary

We summarize our main findings as follows:

- In sudden drop detection, random forest regression with trend removal achieves high PRAUC using different features, although some simple statistical modeling algorithms such as EWMA and WMA can also achieve high PRAUC.
- In correlation change detection, Huber regression without trend removal achieves the highest PRAUC across all predictors.
- Trend removal improves detection accuracy in sudden drop detection across all predictors, while its accuracy varies across predictors in correlation change detection.
- CELLPAD achieves higher F1-Score than TWITTER in sudden drop detection (note that TWITTER currently does not support correlation change detection).
- Retraining the predictor with normal instances only improves PRAUC in correlation change detection.
- CELLPAD remains robust for different choices of thresholds in anomaly detection.

V. RELATED WORK

In this section, we review related work on performance characterization and anomaly detection specifically in the context of cellular networks.

Performance characterization: Several measurement studies analyze real-world traffic traces collected at the cellular network core. Most studies focus on production 3G UMTS cellular networks. For example, Qian *et al.* [30] characterize the cellular network state machine and analyze how control parameters affect radio resource usage and mobile devices' energy consumption. He *et al.* [18] and Qian *et al.* [31] study the interactions between cellular data traffic and signaling overhead. Chen *et al.* [10] uses the supervised regression approach RuleFit [16] to how the round-trip time and loss rates are influenced by different factors such as traffic load and application types. Shafiq *et al.* [32] study the performance degradations in two crowded events. Given the emergence of 4G LTE, Huang *et al.* [20] study the TCP performance based on 10-day traffic traces collected in an LTE network and identify the limitations of TCP over LTE. Our work also analyzes real-world traces based on the measurements at the network core, with specific emphasis on anomaly detection.

Anomaly detection: Some measurement studies pay special attention to anomaly detection in cellular networks. For example, Theera-Ampornpunt *et al.* [34] use classification models to predict network drops and drop duration. Chen *et al.* [11] use customer care calls to infer anomalies through regression. Ahmed *et al.* [3] infer end-to-end performance degradations in four aspects: user locations, content providers, device types, and application types, and their inference models build on robust regression and associative mining. Casas *et al.* [8] apply decision-tree-based classification for anomaly detection, and specifically focus on DNS query performance.

Prior studies perform anomaly detection based on cellular KPIs as in our work. Ciocarlie et al. [13] propose an adaptive

ensemble learning method to address concept drifts in cell anomaly detection. Some studies [4], [14], [23], [27], [33] present automated diagnosis to further identify the root causes of detected KPI anomalies. Chernogorov et al. [12] propose a data mining approach to detect unavailable cells that do not trigger alarms. Besides cellular network management, Twitter [2], [36] proposes an anomaly detection framework for long-term time-series data by addressing seasonality and trend components, yet our evaluation shows that it cannot achieve high detection accuracy as in CELLPAD based on our KPI dataset. Opprentice [25] focuses on KPI anomaly detection in a global search engine and applies machine learning techniques for anomaly detection. In contrast, CELLPAD focuses on providing a unified framework to detect both sudden drops and correlation changes, while correlation changes are not considered by any previous work.

VI. CONCLUSIONS

We study the problem of detecting performance anomalies in cellular networks, and motivate the problem based on a large-scale real-world KPI dataset collected from an operational LTE network. We present CELLPAD, a unified performance anomaly detection framework for cellular networks. CELLPAD targets two types of anomaly detection problems, namely sudden drop detection and correlation change detection. It has the following design elements: (i) support of various statistical and machine-learning-based regression algorithms, (ii) addressing the seasonality and trend patterns in anomaly detection, and (iii) providing a feedback loop for prediction model retraining. Our trace-driven evaluation demonstrates how CELLPAD achieves automated and accurate anomaly detection.

ACKNOWLEDGMENTS

This work was supported by Research Grants Council of Hong Kong (GRF 14204017) and National Natural Science Foundation of China (61572278). This work was done while Jun Wu was visiting the Chinese University of Hong Kong.

REFERENCES

[1] scikit-learn. http://scikit-learn.org/.
[2] Twitter: AnomalyDetection R package. https://github.com/twitter/AnomalyDetection.
[3] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, and H. Yan. Detecting and Localizing End-to-End Performance Degradation for Cellular Data Services. In *Proc. of IEEE INFOCOM*, 2016.
[4] R. Barco, V. Wille, and L. Diez. System for Automated Diagnosis in Cellular Networks Based on Performance Indicators. *European Trans. on Telecommunications*, 16(5):399–409, 2005.
[5] S. Bolton and C. Bo. Linear regression and correlation. *Nurse Anesthesia*, 1990.
[6] L. Breiman. Random Forests. *Machine Learning*, 45(1):5, 2001.
[7] L. I. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees (CART). *Biometrics*, 2015.
[8] P. Casas, P. Fiadino, and A. D'Alconzo. Machine-learning Based Approaches for Anomaly Detection and Classification in Cellular Networks. In *Proc. of IFIP TMA*, 2016.
[9] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):15, 2009.
[10] Y. Chen, N. Duffield, P. Haffner, W.-L. Hsu, G. Jacobson, Y. Jin, S. Sen, S. Venkataraman, and Z.-L. Zhang. Understanding the Complexity of 3G UMTS Network Performance. In *Proc. of IFIP Networking*, 2013.
[11] Y.-C. Chen, G. M. Lee, N. Duffield, L. Qiu, and J. Wang. Event Detection Using Customer Care Calls. In *Proc. of IEEE INFOCOM*, 2013.
[12] F. Chernogorov, S. Chernov, K. Brigatti, and T. Ristaniemi. Sequence-based Detection of Sleeping Cell Failures in Mobile Networks. *Wireless Networks*, 22(6):2029–2048, 2016.
[13] G. Ciocarlie, U. Lindqvist, K. Nitz, S. Nováczki, and H. Sanneck. On the Feasibility of Deploying Cell Anomaly Detection in Operational Cellular Networks. In *Proc. of NOMS*, 2014.
[14] G. F. Ciocarlie, C. Connolly, C.-C. Cheng, U. Lindqvist, S. Nováczki, H. Sanneck, and M. Naseer-ul Islam. Anomaly Detection and Diagnosis for Automatic Radio Network Verification. In *Proc. of MONAMI*, 2014.
[15] J. Davis and M. Goadrich. The Relationship Between Precision-Recall and ROC Curves. In *Proc. of ICML*, 2006.
[16] J. H. Friedman and B. E. Popescu. Predictive Learning via Rule Ensembles. *The Annals of Applied Statistics*, 2008.
[17] H. He and E. A. Garcia. Learning from Imbalanced Data. *IEEE Trans. on Knowledge and Data Engineering*, 21(9):1263–1284, Sep 2009.
[18] X. He, P. P. C. Lee, L. Pan, C. He, and J. C. S. Lui. A Panoramic View of 3G Data/Control-Plane Traffic: Mobile Device Perspective. In *Proc. of IFIP Networking*, 2012.
[19] C. C. Holt. Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages. *International Journal of Forecasting*, 20(1):5–10, 2004.
[20] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proc. of ACM SIGCOMM*, 2013.
[21] P. J. Huber. *Robust Statistics*. Wiley-Interscience, 2011.
[22] M. G. Kendall and A. Stuart. The Advanced Theory of Statistics. Vol.3: Design and Analysis, and Time-series. *Journal of the Royal Statistical Society*, 1983.
[23] R. M. Khanafer, B. Solana, J. Triola, R. Barco, L. Moltsen, Z. Altman, and P. Lazaro. Automated Diagnosis for UMTS Networks Using Bayesian Network Approach. *IEEE Trans. on Vehicular Technology*, 57(4):2451–2461, 2008.
[24] N. Laptev, S. Amizadeh, and I. Flint. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *Proc. of ACM SIGKDD*, 2015.
[25] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng. Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning. In *Proc. of ACM IMC*, 2015.
[26] E. Lundquist. Implement Additive and Multiplicative Holt-Winters Time Series Forecasting Algorithm. https://github.com/etlundquist/holtwint.
[27] S. Nováczki. An Improved Anomaly Detection and Diagnosis Framework for Mobile Network Operators. In *Proc. of DRCN*, 2013.
[28] S. Papadimitriou, J. Sun, and S. Y. Philip. Local Correlation Tracking in Time Series. In *Proc. of IEEE ICDM*, 2006.
[29] B. Pfaff. *Weighted Moving Average*. Springer US, 2001.
[30] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *Proc. of ACM IMC*, 2010.
[31] L. Qian, E. W. W. Chan, P. P. C. Lee, and C. He. Characterization of 3G Control-Plane Signaling Overhead from a Data-Plane Perspective. In *Proc. of ACM MSWiM*, 2012.
[32] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, S. Venkataraman, and J. Wang. Characterizing and Optimizing Cellular Network Performance During Crowded Events. *IEEE/ACM Trans. on Networking*, 24(3):1308–1321, Jun 2016.
[33] P. Szilagyi and S. Nováczki. An Automatic Detection and Diagnosis Framework for Mobile Communication Systems. *IEEE Trans. on Network and Service Management*, 9(2):184–197, 2012.
[34] N. Theera-Ampornpunt, S. Bagchi, K. R. Joshi, and R. K. Panta. Using Big Data for More Dependability: A Cellular Network Tale. In *Proc. of HotDep*, 2013.
[35] P. Tune and M. Roughan. Internet Traffic Matrices: A Primer. In *Recent Advances in Networking*. ACM SIGCOMM, 2013.
[36] O. Vallis, J. Hochenbaum, and A. Kejariwal. A Novel Technique for Long-Term Anomaly Detection in the Cloud. In *Proc. of USENIX HotCloud*, 2014.
[37] P. R. Winters. Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 1960.

An $M:N$ Shared Regenerator Protection Scheme in Translucent WDM Networks

Elias A. Doumith
TICKET Lab - Antonine University
B.P. 40016 Hadat-Baabda - Lebanon
Email: elias.doumith@ua.edu.lb

Sawsan Al Zahr
LTCI - Telecom ParisTech - Université Paris-Saclay
46, rue Barrault F 75634 Paris Cedex 13 - France
Email: sawsan.alzahr@telecom-paristech.fr

***Abstract*—Most studies addressing translucent network design targeted a tradeoff between minimizing the number of deployed regenerators and minimizing the number of regeneration nodes. The latter highly depends on the carrier's strategy and is motivated by various considerations such as power consumption, maintenance and supervision costs. However, concentrating regenerators into a small number of nodes exposes the network to a high risk of data loss in the eventual case of regenerator pool failure. In this paper, we address the problem of survivable translucent network design taking into account the simultaneous effect of four transmission impairments. We propose an exact approach based on a mathematical formulation to solve the problem of regenerator placement while ensuring the network survivability in the hazardous event of a regenerator pool failure. For this purpose, for each accepted request requiring regeneration, we determine several routing paths along with associated valid wavelengths going through different regeneration nodes. In doing so, we implement an $M:N$ shared regenerator protection scheme. Simulation results highlight the gain obtained by reducing the number of regeneration nodes without sacrificing network survivability.**

I. Introduction

The demands for higher bandwidth at lower cost is increasing substantially in today's communication networks. End-users are using more applications that require reliable connectivity and faster data transfer. This constant growth of broadband services is pushing service providers and equipment vendors to look for scalable optical networks. However, the major limitations to scalability are the optical signal reach and the power consumption. Indeed, transmission impairments induced by long-haul optical equipment may significantly degrade the quality of the optical signal [1], [2]. In order to compensate for the signal degradation and to extend the signal reach, 3R (reamplification, reshaping, and retiming) regenerators may be deployed in the network which will increase the deployment cost as well as the power consumption. Thus, the only viable solution for most wide-area networks is the translucent approach where a small number of regenerators will be deployed in the network.

Since the early 2000s, many researches have addressed the optimal design of translucent optical networks. These studies highlighted that an intelligent deployment of the 3R regenerators allows a translucent optical network to perform similarly to a fully opaque network [3], [4]. The early studies in this field tried to minimize the number of regenerators based on the network topology (such as the number of neighbors or the average distance to neighbors). Indeed, in [4]–[7], a few number of nodes, with the highest number of pre-computed shortest paths traversing them, are selected as regeneration nodes. Afterward, regenerators are assigned to traffic requests based on a QoT evaluation method. In contrast with [4]–[6], additional regeneration nodes may be added during the routing and wavelength assignment process in order to maximize the number of satisfied requests while minimizing the number of regenerators and regeneration nodes [7]. However, these approaches are not realistic as the regenerator placement does not account for signal degradation. Other studies targeted the minimization of the number of regenerators in order to satisfy a given static traffic matrix [3], [8]–[11]. It was until 2011 that the problem of regenerator placement has been first investigated under dynamic but deterministic traffic model [12]. Under such a traffic pattern, one can take advantage from the dynamics of the traffic model so that deployed regenerators may be shared among multiple time-disjoint requests.

In the early 2000s, all studies that addressed the regenerator placement problem were based on empirical laws or heuristic approaches [3], [8]. It was until 2008 that Pan *et al.* proposed in [9] the first exact approach for regenerator placement. The aim was to minimize the number of regeneration nodes under static traffic requests with $1+1$ protection scheme. The QoT constraint was considered as a maximum optical reach. Later on, two exact approaches were proposed taking into account different linear and nonlinear impairments [10], [11]. In these studies, the problem of regenerator placement was formulated as a virtual topology design problem where the QoT constraint was considered as a minimum admissible Q-factor. In [10], the network topology is represented by an equivalent graph where two non-adjacent nodes, interconnected by a path with an admissible Q-factor, are connected by a crossover edge in the equivalent graph. In [11], a set of pre-computed paths is selected and regenerators are deployed along these paths. By routing a set of static traffic requests on the virtual topology, the aim of the proposed approaches is to minimize the number of regenerators or the number of regeneration nodes.

In our previous works [12], [13], we proposed an exact formulation for the problem of translucent network design in order to determine the optimal number of regenerators and

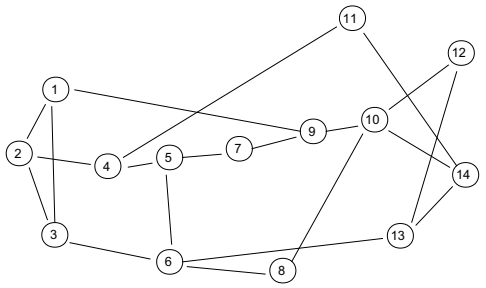
ISBN 978-3-903176-08-9

regeneration nodes. Instead of considering permanent/semi-permanent traffic requests commonly used for network design purposes, we considered a more generic and realistic traffic model referred to as Scheduled Lightpath Demands (SLDs). Such a model allows us to easily represent the dynamism of the traffic without losing its deterministic aspect required in order to design the network. In [14], we extended our work to account for the traffic forecast uncertainty. More precisely, instead of designing a translucent network that can accommodate a single set of traffic requests, we considered in the design phase different sets of traffic requests that correspond to the traffic forecasts at different epochs in the future. The resulting optimal translucent optical network design can be used to accommodate any of the considered traffic forecasts.

In all the previous works, researchers tried to achieve the optimal translucent network design motivated by restrictions on capital and operational expenditures (CapEx/OpEx). Such a design consists in achieving a trade-off between minimizing the number of regenerators and minimizing the number of regeneration nodes. As the number of deployed regenerators has been reduced to a minimum, their utilization ratio is relatively high exposing the network to a high risk of data loss in the eventual case of regenerator pool failure. Indeed, if a regenerator pool fails, most of the traffic requests that were planned to be regenerated at this node cannot be regenerated at the other operational regenerator pools because of their high utilization ratio. Subsequently, the impacted traffic requests will suffer from excessive optical signal degradation and their data will be lost. In order to ensure high network availability, the network typically has redundant hardware that makes it available despite failures. The same analysis is applicable if we investigate the failure of multi-channel all-optical regenerators instead of electrical regenerators pool [15].

In this paper, we target to achieve the optimal translucent network design taking into account both operational and backup regenerators. For this purpose, for each accepted request requiring regeneration, we determine several routing paths along with associated valid wavelengths going through different regeneration nodes. In doing so, we implement an $M:N$ shared regenerator protection scheme minimizing the number of regenerators and regeneration nodes without sacrificing network survivability. In order to shorten the time needed to reach the optimal regenerator deployment solution, we tighten the formulation by adding constraints that will help discard equivalent solutions without omitting the optimal solution. As for the QoT constraint, we consider a minimum admissible Q-factor reflecting the simultaneous effect of four well known transmission impairments, namely chromatic dispersion, polarization mode dispersion, amplified spontaneous emission, and self phase modulation.

The remainder of this paper is organized as follows. In Section II, we present a description of the investigated scenarios. Our approach of survivable translucent network design is provided in Section III followed in Section IV by an analysis of the numerical results. Finally, we draw our conclusions in Section V.



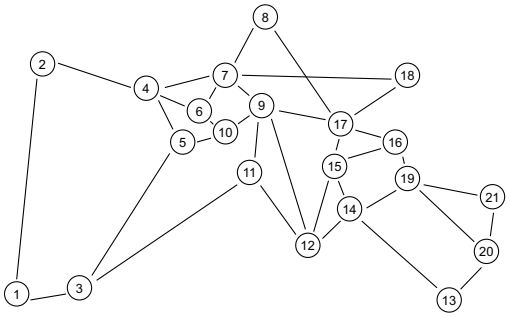
Links	Dist. (km)
u_1-u_2	480
u_1-u_3	680
u_1-u_9	1500
u_2-u_3	480
u_2-u_4	680
u_3-u_6	850
u_4-u_5	300
u_4-u_{11}	1500
u_5-u_6	480
u_5-u_7	400
u_6-u_8	850
u_6-u_{13}	1500
u_7-u_9	400
u_8-u_{10}	620
u_9-u_{10}	400
u_{10}-u_{12}	480
u_{13}-u_{14}	400

Links	Dist. (km)
u_{10}-u_{14}	680

Links	Dist. (km)
u_{11}-u_{14}	400

Links	Dist. (km)
u_{12}-u_{13}	680

Fig. 1. The north American 14-node 20-link NSF backbone network.



Links	Dist. (km)
u_1-u_2	1730
u_1-u_3	530
u_2-u_4	490
u_3-u_5	1110
u_3-u_{11}	1210
u_4-u_5	360
u_4-u_6	340
u_4-u_7	380
u_5-u_{10}	300
u_6-u_7	180
u_6-u_{10}	190
u_7-u_8	650
u_7-u_9	380
u_7-u_{18}	1150
u_8-u_{17}	670
u_9-u_{10}	210
u_9-u_{11}	400
u_{12}-u_{14}	540
u_{14}-u_{15}	280
u_{16}-u_{17}	310
u_{19}-u_{21}	680
u_{20}-u_{21}	310

Links	Dist. (km)
u_9-u_{12}	1010
u_{12}-u_{15}	800
u_{14}-u_{19}	310
u_{16}-u_{19}	170

Links	Dist. (km)
u_9-u_{17}	430
u_{13}-u_{14}	1140
u_{15}-u_{16}	60
u_{17}-u_{18}	540

Links	Dist. (km)
u_{11}-u_{12}	730
u_{13}-u_{20}	550
u_{15}-u_{17}	260
u_{19}-u_{20}	660

Fig. 2. The European 21-node 34-link EBN backbone network.

II. Investigated Scenarios

A. Network Environment

A network node is modeled as an optical cross-connect (OXC) consisting of several wavelength selective switches (WSSs) and an access unit responsible for adding/dropping traffic requests [3], [16]. A limited number of network nodes are equipped with a pool of regenerators. A regenerator is a tunable transmitter-receiver pair along with a processing unit responsible for re-amplifying, re-shaping, and re-timing the optical signal in the electrical domain. A network link is composed of two unidirectional standard single mode fibers (one SMF in each direction) carrying each $W = 20$ wavelengths in the C-band. In order to compensate for the attenuation and the chromatic dispersion, double stage Erbium-doped fiber amplifiers (EDFAs) are deployed every 80 km along with dispersion compensating fibers (DCFs). Furthermore, inline optical gain equalizers are deployed every 400 km. For the numerical evaluation, we consider the 14-node 20-link NSF network (Figure 1) as well as the 21-node 34-link EBN network (Figure 2). Table I summarizes the parameters of all the equipment deployed in the network.

Transmission impairments induced by long-haul optical equipment accumulate along lightpaths and may significantly degrade the quality of the optical signal. We distinguish between two types of impairments, namely linear and non-

TABLE I
TRANSMISSION SYSTEM PARAMETERS

Parameter	Value
Number of wavelengths	20
Wavelengths (nm)	$1538.97 - 1554.13$
Channel spacing (GHz)	100
Channel bit rate (Gbps)	10
SMF input power (dBm)	-1
SMF loss (dB/km)	0.23

Parameter	Value
SMF PMD (ps/$\sqrt{km}$)	0.1
SMF dispersion (ps/nm.km)	$\overset{av.}{=} 17$[1]
DCF input power (dBm)	-7
DCF loss (dB/km)	0.6
DCF dispersion (ps/nm.km)	$\overset{av.}{=} -90$[1]
DCF PMD (ps/$\sqrt{km}$)	0.08

Parameter	Value
Switching loss (dB)	-13
Inline EDFA Noise Figure (dB)	$\overset{av.}{=} 6$[1]
Booster EDFA Noise Figure (dB)	$\overset{av.}{=} 5.25$[1]
Pre-compensation (ps)	-800
Dispersion slope $(ps/nm/span)$	100
Q-factor threshold (dB)	15.6

[1] It is only the mean value; the real value depends on the selected wavelength value.

linear impairments. Linear impairments are proportional to the traveled distance and depend on the signal itself (*e.g.,* chromatic dispersion CD, polarization mode dispersion PMD, amplified spontaneous emission ASE), while nonlinear impairments arise from the signal itself and from the interaction between neighboring channels (*e.g.,* self-phase modulation SPM, cross-phase modulation XPM, four wave mixing FWM) [17]. Various metrics can be used to evaluate the signal quality at the end of a lightpath. Among these metrics, the bit error rate (BER) is the most appropriate criterion because it aggregates the effects of all physical impairments. In this paper, we make use of "*BER-Predictor*" previously introduced in [18] to estimate the BER value at the end of each operational lightpath. BER-Predictor computes the Q-factor as a function of the penalties simultaneously induced by four physical impairments, namely ASE, CD, PMD, and SPM. The analytical relation between the Q-factor and the aforementioned impairments has been derived from analytical formulas and experimental measurements [2]. BER-Predictor can be used assuming either flat or non-flat spectral responses of optical equipment. A flat transmission system behaves the same regardless the wavelength value, while in a non-flat transmission system, the impairments induced by some equipment such as fibers and amplifiers depend on the wavelength value.

B. Traffic Model

In this paper, we make use of the SLD traffic model that allows us to capture the long-term aspect of the traffic as well as its dynamism. The i^{th} SLD request δ_i is represented by the tuple $(\mathfrak{s}_i, \mathfrak{r}_i, \alpha_i, \beta_i)$. The source node $\mathfrak{s}_i$ and the destination node $\mathfrak{r}_i$ of a request are chosen uniformly among the network nodes such that there is no demand between two adjacent nodes. The idea is to exclude one-hop lightpaths that do not require any regeneration. The attributes α_i and β_i denote the set-up and tear-down dates of a request. We first assume that all the requests arrive at the same time ($\alpha_i = 0,\ \forall i$) and, if accepted, will hold the network for the whole simulation period ($\beta_i = \Delta,\ \forall i$). Such requests are known as permanent lightpath demands (PLDs). Without changing the source and destination nodes of the requests, we then reduce the period where they are active according to a parameter π ($0 < \pi \leqslant 1$). More precisely, the activity period $(\beta_i - \alpha_i)$ of a request δ_i is chosen uniformly in the interval $[\Delta \times \pi - 1, \Delta \times \pi + 1]$, and the set-up date α_i is chosen randomly while ensuring that δ_i still ends before the expiration of the simulation period ($\beta_i \leqslant \Delta$).

III. $M:N$ SHARED REGENERATOR PROTECTION SCHEME

We target to achieve the optimal translucent network design taking into account both operational and backup regenerators. These regenerators are required in order to cope with transmission impairments and for wavelength conversion needs. Given the network topology and the set of traffic requests, we seek to maximize the number of accepted requests. For each accepted request requiring regeneration, we determine several routing paths along with associated valid wavelengths going through different regeneration nodes. In doing so, we implement an $M:N$ shared regenerator protection scheme minimizing the number of regenerators and regeneration nodes without sacrificing network survivability. The optimal translucent network design is achieved by formulating the problem as a Mixed-Integer Linear Program and solving it using traditional solvers.

In order to improve the scalability of our approach, we decompose the problem into the "Routing and Regenerator Placement" (RRP) sub-problem and the "Wavelength Assignment and Regenerator Placement" (WARP) sub-problem. In the former, we place regenerators and route the traffic requests while assuming that the QoT is independent of the wavelength value. In the latter, additional regenerators may be required to overcome the dependency of the QoT on the wavelength value. Deployed regenerators may be shared among multiple non-concurrent requests. The common parameters for these two sub-problems are:

- The network topology represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{\mathfrak{u}_v, v = 1 \cdots N\}$ is the set of network nodes and $\mathcal{E} = \{\mathfrak{e}_e = (\mathfrak{u}_v, \mathfrak{u}_u) \in \mathcal{V} \times \mathcal{V}, e = 1 \cdots L\}$ is the set of unidirectional fiber-links connecting these nodes.
- The set of available wavelengths $\Lambda = \{\lambda_\ell, \ell = 1 \cdots W\}$ on each fiber-link in the network.
- The threshold Q_{th} for an admissible Q-factor.

As we are concerned by the failure of a regenerator pool that can be located at any node of the network, we consider, for each of the sub-problems, $N+1$ different scenarios. Scenario '0' corresponds to the case where all the regenerator pools are fully operational, while scenario 's' ($s = 1 \cdots N$) corresponds to the case where the regenerator pool at node $\mathfrak{u}_s$ is down.

A. Routing and Regenerator Placement

In this sub-problem, we assume that the QoT is independent of the wavelength value. In other words, the QoT of a lightpath transmitted over a wavelength λ_ℓ is the same as if the lightpath

was transmitted over the reference wavelength $\lambda_c = 1550$ nm. This is obtained by setting BER-predictor to operate under the flat spectral response configuration. The RRP sub-problem is formulated as follows:

1) Parameters:

• The set of traffic requests $\mathcal{D} = \{\delta_i, i = 1 \cdots D\}$. Each request δ_i is represented by a tuple $(\mathfrak{s}_i \in \mathcal{V}, \mathfrak{r}_i \in \mathcal{V}, \alpha_i, \beta_i)$.

• The ordered set $\mathcal{T}$ grouping the set-up and tear-down dates of all the requests in $\mathcal{D}$.

$$\mathcal{T} = \bigcup_{\delta_i \in \mathcal{D}} \{\alpha_i, \beta_i\} = \{\tau_1, \cdots, \tau_{\mathfrak{T}}\} \tag{1}$$

such that $\tau_1 < \tau_2 < \cdots < \tau_{\mathfrak{T}}$ and $\mathfrak{T} = |\mathcal{T}|$

• The request matrix $\Theta = \{\theta_{i,t}, i = 1 \cdots D, t = 1 \cdots \mathfrak{T}\}$ representing the traffic requests over time. An element $\theta_{i,t}$ of this matrix is a binary value specifying the presence ($\theta_{i,t} = 1$) or the absence ($\theta_{i,t} = 0$) of request δ_i at time instant τ_t.

$$\theta_{i,t} = \begin{cases} 1 & \text{if } \alpha_i \leqslant \tau_t < \beta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

• For each request δ_i, we compute K-shortest paths in terms of real length (*cf.* Figures 1 and 2) connecting its source node $\mathfrak{s}_i$ to its destination node $\mathfrak{r}_i$. Let $\mathcal{P}_i = \{p_{i,j}, j = 1 \cdots K\}$ be the set of available shortest paths for request δ_i. The j^{th}-shortest path $p_{i,j}$ of δ_i is the ordered set of unidirectional links $\{\mathfrak{e}_{e_1}, \mathfrak{e}_{e_2}, \cdots, \mathfrak{e}_{e_{|p_{i,j}|}}\}$ traversed in the source-destination direction ($\mathfrak{s}_i \mapsto \mathfrak{r}_i$).

• For each link pair $(\mathfrak{e}_m, \mathfrak{e}_n)$ along a path $p_{i,j}$, we compute by means of BER-Predictor the Q-factor value $\mathfrak{Q}_{i,j}^{m,n}$ of the directed path-segment delimited by the *source* node of link $\mathfrak{e}_m$ and the *destination* node of link $\mathfrak{e}_n$ ($\mathfrak{e}_m \preccurlyeq \mathfrak{e}_n$). As we assumed a flat spectral response, $\mathfrak{Q}_{i,j}^{m,n}$ is constant for all the wavelengths along this directed path-segment.

2) Variables:

• The binary acceptance variables $\mathfrak{a}_i$, $i = 1 \cdots D$.
$\mathfrak{a}_i = 1$, if request δ_i is accepted. $\mathfrak{a}_i = 0$, otherwise.

• The binary variables $\mathfrak{p}_{s,i,j}$, $s = 0 \cdots N$, $i = 1 \cdots D$, $j = 1 \cdots K$.
$\mathfrak{p}_{s,i,j} = 1$, if in scenario 's', request δ_i is routed over the j^{th}-shortest path between $\mathfrak{s}_i$ and $\mathfrak{r}_i$. $\mathfrak{p}_{s,i,j} = 0$, otherwise.

• The binary variables $\zeta_{s,i,j}^{m,n}$, $s = 0 \cdots N$, $i = 1 \cdots D$, $j = 1 \cdots K$, $m = 1 \cdots L$, $n = 1 \cdots L$.
$\zeta_{s,i,j}^{m,n}$ is an intermediate variable used to insure that the Q-factor at the end of the directed path-segment delimited by the source node of link $\mathfrak{e}_m$ and the destination node of link $\mathfrak{e}_n$ along the j^{th}-shortest path $p_{i,j}$ used by the request δ_i in scenario 's' exceeds the predefined threshold.

• The binary variables $\mathfrak{d}_{s,i,u}$, $s = 0 \cdots N$, $i = 1 \cdots D$, $u = 1 \cdots N$.
$\mathfrak{d}_{s,i,u} = 1$, if in scenario 's', request δ_i is regenerated at node $\mathfrak{u}_u$. $\mathfrak{d}_{s,i,u} = 0$, otherwise.

• The non-negative integer variables $\psi_{s,u,t}$, $s = 0 \cdots N$, $u = 1 \cdots N$, $t = 1 \cdots \mathfrak{T}$.
$\psi_{s,u,t}$ is equal to the number of regenerators that are in use in scenario 's' at node $\mathfrak{u}_u$ and time instant τ_t.

• The binary variables ϕ_u, $u = 1 \cdots N$.
$\phi_u = 1$, if node $\mathfrak{u}_u$ is selected as a regeneration node. $\phi_u = 0$, otherwise.

• The non-negative integer variables $\mathfrak{R}_u$, $u = 1 \cdots N$.
$\mathfrak{R}_u$ denotes the number of regenerators deployed at node $\mathfrak{u}_u$.

3) Constraints:

• If request δ_i is accepted, it is routed over a single path among the available K-shortest paths between $\mathfrak{s}_i$ and $\mathfrak{r}_i$ in each of the considered scenarios. $\forall s = 0 \cdots N$, $\forall i = 1 \cdots D$,

$$\sum_{j=1 \cdots K} \mathfrak{p}_{s,i,j} = \mathfrak{a}_i \tag{3}$$

• In each scenario 's', the number of requests routed over a single fiber-link $\mathfrak{e}_m$ must not exceed, at any time, the number of wavelengths on that fiber-link. $\forall s = 0 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$, $\forall m = 1 \cdots L$,

$$\sum_{i=1 \cdots D} \theta_{i,t} \times \sum_{j=1 \cdots K \setminus \mathfrak{e}_m \in p_{i,j}} \mathfrak{p}_{s,i,j} \leqslant W \tag{4}$$

• In each scenario 's', the Q-factor at the end of the path-segment delimited by any two distinct nodes along the selected path of an accepted request must exceed the predefined threshold Q_{th}. Otherwise, regenerators must be deployed at some intermediate nodes along this path-segment. This can be expressed mathematically as follows: $\forall s = 0 \cdots N$, $\forall i = 1 \cdots D$, $\forall j = 1 \cdots K$, $\forall \mathfrak{e}_n \in p_{i,j}$,

$$\sum_{\mathfrak{e}_m \in p_{i,j} \setminus \mathfrak{e}_m \preccurlyeq \mathfrak{e}_n} \zeta_{s,i,j}^{m,n} \times \mathfrak{Q}_{i,j}^{m,n} \quad \geqslant \quad \mathfrak{p}_{s,i,j} \times Q_{th} \tag{5a}$$

$$\sum_{\mathfrak{e}_m \in p_{i,j} \setminus \mathfrak{e}_m \preccurlyeq \mathfrak{e}_n} \zeta_{s,i,j}^{m,n} \quad = \quad \mathfrak{p}_{s,i,j} \tag{5b}$$

• By collecting all the previous constraints on the variables $\zeta_{s,i,j}^{m,n}$, we can determine, for each scenario 's', all the intermediate nodes $\mathfrak{u}_u$ where request δ_i should be regenerated (except at its source node $\mathfrak{s}_i$). $\forall s = 0 \cdots N$, $\forall i = 1 \cdots D$, $\forall j = 1 \cdots K$, $\forall \mathfrak{e}_m = (\mathfrak{u}_u, \mathfrak{u}_v) \in p_{i,j}$, $\forall \mathfrak{e}_n \in p_{i,j}$ such that $\mathfrak{e}_m \preccurlyeq \mathfrak{e}_n$ and $\mathfrak{u}_u \neq \mathfrak{s}_i$,

$$\mathfrak{d}_{s,i,u} \geqslant \zeta_{s,i,j}^{m,n} \tag{6}$$

• In each scenario 's', the number of regenerators $\psi_{s,u,t}$ in use at node $\mathfrak{u}_u$ and time instant τ_t can then be computed as: $\forall s = 0 \cdots N$, $\forall u = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\psi_{s,u,t} = \sum_{i=1 \cdots D} \theta_{i,t} \times \mathfrak{d}_{s,i,u} \tag{7}$$

• The number of regenerators $\mathfrak{R}_u$ deployed at node $\mathfrak{u}_u$ is the maximum number of regenerators that are in use at any time for all the considered scenarios. $\forall s = 0 \cdots N$, $\forall u = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\mathfrak{R}_u \geqslant \psi_{s,u,t} \tag{8}$$

• A node is considered as a regeneration node if it hosts at least a single regenerator. $\forall u = 1 \cdots N$,

$$\phi_u \geqslant 10^{-3} \times \mathfrak{R}_u \tag{9}$$

• Finally, regenerator pool failure at node $\mathfrak{u}_s$ is simulated by setting to zero the number of regenerators that can be deployed at this node in its corresponding scenario. $\forall s = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\psi_{s,s,t} = 0 \tag{10}$$

4) Objective: The objective of the RRP sub-problem is to maximize the number of accepted requests while minimizing the number of regenerators and regeneration nodes. This objective is expressed as:

$$\max \quad \gamma_1 \times \sum_{i=1\cdots D} \mathfrak{a}_i - \gamma_2 \times \sum_{u=1\cdots N} \phi_u - \gamma_3 \times \sum_{u=1\cdots N} \mathfrak{R}_u \qquad (11)$$

where γ_1, γ_2, and γ_3 are three non-negative real numbers used to stress the regenerators concentration into a limited number of regeneration nodes, the minimization of the required number of regenerators, the maximization of the number of accepted requests, or any combination of the previous objectives.

5) Performance Improvement: Although the previous formulation is correct, the feasible solution space is quite large. In order to shorten the time needed to solve the RRP sub-problem, we reduce the solution space while paying attention to not omit the optimal solution. This is achieved by cutting regions of the solution space that do not contain any improvement. Indeed, if we notice that when node $\mathfrak{u}_s$ is not selected as a regeneration node, the scenario 's' representing the failure of the regenerator pool at this node is obvious as it should not affect the accepted requests nor their associated paths. More precisely, the paths assigned to the requests in scenario 's' should be identical to the paths obtained in scenario '0'. This is obtained by replacing Equation (3) with the following:

- If request δ_i is accepted in scenario '0' (no regenerator pool failure), it is routed over a single path $p_{i,j}$ among the available K-shortest paths between $\mathfrak{s}_i$ and $\mathfrak{r}_i$. $\forall i = 1\cdots D$,

$$\sum_{j=1\cdots K} \mathfrak{p}_{0,i,j} = \mathfrak{a}_i \qquad (12)$$

- For each failure scenario 's', if node $\mathfrak{u}_s$ is a regeneration node ($\phi_s = 1$), we select a single path $p_{i,j}$ for each accepted request δ_i. Conversely, if node $\mathfrak{u}_s$ is not a regeneration node ($\phi_s = 0$), we set all the variables $\mathfrak{p}_{s,i,j}$ to zero. In this way, we do not assign any path to the accepted requests. Once we obtain the optimal solution, we route, in a post-processing step, each accepted request in scenario 's' on the same path as in scenario '0'. This is expressed mathematically as: $\forall s = 1\cdots N$, $\forall i = 1\cdots D$,

$$\sum_{j=1\cdots K} \mathfrak{p}_{s,i,j} = \mathfrak{a}_i \times \phi_s \qquad (13)$$

The expression $\mathfrak{a}_i \times \phi_s$ is non-linear since it is the product of two binary variables. However, this product can be linearized by means of additional constraints. Thus, Equation (13) can be written in linear form as follows: $\forall s = 1\cdots N$, $\forall i = 1\cdots D$,

$$\sum_{j=1\cdots K} \mathfrak{p}_{s,i,j} \leq \mathfrak{a}_i \qquad (14a)$$

$$\sum_{j=1\cdots K} \mathfrak{p}_{s,i,j} \leq \phi_s \qquad (14b)$$

$$\sum_{j=1\cdots K} \mathfrak{p}_{s,i,j} \geq \mathfrak{a}_i + \phi_s - 1 \qquad (14c)$$

B. Wavelength Assignment and Regenerator Placement

In the solution obtained at the end of the RRP sub-problem, some requests are accepted; others are rejected. Rejected requests are definitely dropped and removed from the problem. Let $\widehat{\mathcal{D}} = \{\widehat{\delta}_i, i = 1\cdots\widehat{D}\}$ be the set of accepted requests. Each accepted request has been assigned a single path between its source and its destination nodes in the normal operational scenario ($s = 0$) as well as in each considered failure scenario ($s = 1\cdots N$). This request may have been regenerated at some intermediate nodes along its path. Without altering its selected path, an accepted request requiring regeneration in a particular scenario is divided into path-segments whenever it passes through its regeneration node. As the routes and the regenerators assigned to a given request may vary from one scenario to another, its decomposition into sub-paths will also vary. Let $\widetilde{\mathcal{D}}_s = \{\widetilde{\delta}_{s,d}, d = 1\cdots\widetilde{D}_s\}$ be the modified sets of accepted requests (one modified set of requests for each considered scenario $s = 0\cdots N$) containing the accepted requests with an admissible QoT (no regeneration required) as well as the path-segments of the accepted requests requiring regeneration.

In the WARP sub-problem, we assign to each request $\widetilde{\delta}_{s,d}$ in a scenario 's' a single continuous wavelength between its source and its destination nodes. When this is not possible, additional regenerators are deployed to serve as wavelength converters. Moreover, all these requests have an acceptable QoT if they are transmitted over the reference wavelength $\lambda_c = 1550$ nm. If a request $\widetilde{\delta}_{s,d}$ is transmitted over another wavelength, its QoT may be degraded due to the non-flat spectral response of optical equipment. This problem can be resolved by deploying additional regenerators at some intermediate nodes along the path assigned to $\widetilde{\delta}_{s,d}$. However, it may happen that the required additional regenerator for a request $\widetilde{\delta}_{s,d}$ in scenario 's' needs to be deployed at node $\mathfrak{u}_s$. Recalling that scenario 's' corresponds to the case where the regenerator pool at node $\mathfrak{u}_s$ is down, no regenerators can be deployed at node $\mathfrak{u}_s$ and the corresponding request will be rejected. In order to optimize the network ressources' utilization, whenever a request $\widetilde{\delta}_{s,d}$ is rejected, we also reject the original request $\widehat{\delta}_i$ and all its path-segments from all the scenarios. Furthermore, we remove all the regenerators that were required by the original request $\widehat{\delta}_i$ in the RRP sub-problem. For this purpose, we define the function $\mathfrak{F}(.)$ that returns, for each request $\widetilde{\delta}_{s,d} \in \widetilde{\mathcal{D}}_s$, the index of the associated original request $\widehat{\delta}_i \in \widehat{\mathcal{D}}$.

$$\mathfrak{F}\left(\widetilde{\delta}_{s,d}\right) = i \qquad (15)$$

Finally, we take advantage of the regenerators deployed in the RRP sub-problem when they are not in use. Hence, the WARP sub-problem is formulated as follows:

1) Parameters:

- The set of original requests $\widehat{\mathcal{D}} = \{\widehat{\delta}_i, i = 1\cdots\widehat{D}\}$ that were accepted in the RRP sub-problem. Each accepted original request $\widehat{\delta}_i$ is represented by a tuple $(\mathfrak{s}_i \in \mathcal{V}, \mathfrak{r}_i \in \mathcal{V}, \alpha_i, \beta_i)$.
- For each considered scenario 's', an accepted original request $\widehat{\delta}_i$ is routed over a single path and may be regenerated at some intermediate nodes along this path. This is captured by the binary parameters $\widehat{\mathfrak{d}}_{s,i,u}$, $s = 0\cdots N$, $i = 1\cdots\widehat{D}$, $u = 1\cdots N$. $\widehat{\mathfrak{d}}_{s,i,u} = 1$, if original request $\widehat{\delta}_i$ was regenerated in scenario 's' of the RRP sub-problem at node $\mathfrak{u}_u$. $\widehat{\mathfrak{d}}_{s,i,u} = 0$, otherwise.
- The modified sets of requests $\widetilde{\mathcal{D}}_s = \{\widetilde{\delta}_{s,d}, d = 1\cdots\widetilde{D}_s\}$ ($s = 0\cdots N$) obtained by dividing the original requests into path-segments at the nodes where they were regenerated.

Each request $\widetilde{\delta}_{s,d}$ is represented by a tuple $(\mathfrak{s}_{s,d} \in \mathcal{V}, \mathfrak{r}_{s,d} \in \mathcal{V}, \alpha_{s,d}, \beta_{s,d})$.

• At the end of the RRP sub-problem, each request $\widetilde{\delta}_{s,d}$ is routed over a single path $p_{s,d}$ represented as the ordered set of unidirectional links $\{\mathfrak{e}_{e_1}, \mathfrak{e}_{e_2}, \cdots, \mathfrak{e}_{e_{|p_{s,d}|}}\}$ traversed in the source-destination direction ($\mathfrak{s}_{s,d} \mapsto \mathfrak{r}_{s,d}$). For each wavelength $\lambda_\ell \in \Lambda$, we compute by means of BER-Predictor the Q-factor value $\mathfrak{Q}^{\ell}_{s,d}$ at the destination node $\mathfrak{r}_{s,d}$ of the selected path $p_{s,d}$. In this sub-problem, BER-predictor operates under the non-flat spectral response configuration.

• The ordered set $\mathcal{T} = \{\tau_t, t = 1 \cdots \mathfrak{T}\}$ grouping the set-up and tear-down dates of all the requests in $\widehat{\mathcal{D}}$ (*cf.* Equation (1)).

• The request matrix $\widehat{\Theta} = \{\widehat{\theta}_{i,t}, i = 1 \cdots \widehat{D}, t = 1 \cdots \mathfrak{T}\}$ representing the original requests $\widehat{\delta}_i$ over time. An element $\widehat{\theta}_{i,t}$ of this matrix is a binary value specifying the presence $(\widehat{\theta}_{i,t} = 1)$ or the absence $(\widehat{\theta}_{i,t} = 0)$ of request $\widehat{\delta}_i$ at time instant τ_t.

$$\widehat{\theta}_{i,t} = \begin{cases} 1 & \text{if } \alpha_i \leqslant \tau_t < \beta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

• For each scenario 's' ($s = 0 \cdots N$), the new request matrix $\widetilde{\Theta}_s = \{\widetilde{\theta}_{s,d,t}, d = 1 \cdots \widetilde{D}_s, t = 1 \cdots \mathfrak{T}\}$ representing the requests $\widetilde{\delta}_{s,d}$ over time. An element $\widetilde{\theta}_{s,d,t}$ of this matrix is a binary value specifying the presence $(\widetilde{\theta}_{s,d,t} = 1)$ or the absence $(\widetilde{\theta}_{s,d,t} = 0)$ of request $\widetilde{\delta}_{s,d}$ at time instant τ_t.

$$\widetilde{\theta}_{s,d,t} = \begin{cases} 1 & \text{if } \alpha_{s,d} \leqslant \tau_t < \beta_{s,d}, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

2) Variables:

• The binary acceptance variables $\mathfrak{a}_i$, $i = 1 \cdots \widehat{D}$.

$\mathfrak{a}_i = 1$, if original request $\widehat{\delta}_i$ is still accepted. $\mathfrak{a}_i = 0$, otherwise.

• The binary variables $\rho^{\ell}_{s,d,m}$, $s = 0 \cdots N$, $d = 1 \cdots \widetilde{D}_s$, $m = 1 \cdots L$, $\ell = 1 \cdots W$.

$\rho^{\ell}_{s,d,m} = 1$, if in scenario 's', request $\widetilde{\delta}_{s,d}$ is transmitted over wavelength λ_ℓ along link $\mathfrak{e}_m$. $\rho^{\ell}_{s,d,m} = 0$, otherwise.

• The binary variables $\mathfrak{d}_{s,d,u}$, $s = 0 \cdots N$, $d = 1 \cdots \widetilde{D}_s$, $u = 1 \cdots N$.

$\mathfrak{d}_{s,d,u} = 1$, if in scenario 's', request $\widetilde{\delta}_{s,d}$ is regenerated in the WARP sub-problem at node $\mathfrak{u}_u$. $\mathfrak{d}_{s,d,u} = 0$, otherwise.

• The non-negative integer variables $\psi_{s,u,t}$, $s = 0 \cdots N$, $u = 1 \cdots N$, $t = 1 \cdots \mathfrak{T}$.

$\psi_{s,u,t}$ is equal to the number of regenerators that are in use in scenario 's' at node $\mathfrak{u}_u$ and time instant τ_t.

• The binary variables ϕ_u, $u = 1 \cdots N$.

$\phi_u = 1$, if node $\mathfrak{u}_u$ is a regeneration node. $\phi_u = 0$, otherwise.

• The non-negative integer variables $\mathfrak{R}_u$, $u = 1 \cdots N$.

$\mathfrak{R}_u$ denotes the total number of regenerators deployed at node $\mathfrak{u}_u$ (including those already deployed in the RRP sub-problem).

3) Constraints:

• If original request $\widehat{\delta}_i$ remains accepted, a single wavelength is reserved on all the links that are traversed by its sub-paths $\widetilde{\delta}_{s,d}$ in all the scenarios. $\forall s = 0 \cdots N$, $\forall d = 1 \cdots \widetilde{D}_s$, $\forall m = 1 \cdots L$,

$$\sum_{\ell=1 \cdots W} \rho^{\ell}_{s,d,m} = \begin{cases} \mathfrak{a}_{\mathfrak{F}\left(\widetilde{\delta}_{s,d}\right)} & \text{if } \mathfrak{e}_m \in p_{s,d}, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

• In each scenario 's', each wavelength on a link can be used at most once at a given time instant. $\forall s = 0 \cdots N$, $\forall m = 1 \cdots L$, $\forall \ell = 1 \cdots W$, $t = 1 \cdots \mathfrak{T}$,

$$\sum_{d=1 \cdots \widetilde{D}_s} \rho^{\ell}_{s,d,m} \times \widetilde{\theta}_{s,d,t} \leqslant 1 \tag{19}$$

• A path $p_{s,d}$ must use the same wavelength on any two consecutive links unless a regenerator is deployed at the node in common to the two links. $\forall s = 0 \cdots N$, $\forall d = 1 \cdots \widetilde{D}_s$, $\forall \ell = 1 \cdots W$, $\forall \mathfrak{e}_m = (\mathfrak{u}_v, \mathfrak{u}_u) \in p_{s,d}$, $\forall \mathfrak{e}_n = (\mathfrak{u}_u, \mathfrak{u}_l) \in p_{s,d}$,

$$\rho^{\ell}_{s,d,m} - \rho^{\ell}_{s,d,n} \leqslant \mathfrak{d}_{s,d,u} \tag{20a}$$

$$\rho^{\ell}_{s,d,n} - \rho^{\ell}_{s,d,m} \leqslant \mathfrak{d}_{s,d,u} \tag{20b}$$

• The Q-factor at the destination node of a request must exceed the predefined threshold Q_{th}. Otherwise, a regenerator is deployed at an intermediate node along the path of the degraded request. $\forall s = 0 \cdots N$, $\forall d = 1 \cdots \widetilde{D}_s$, $\forall \ell = 1 \cdots W$, $\forall \mathfrak{e}_m \in p_{s,d}$,

$$\mathfrak{Q}^{\ell}_{s,d} \times \rho^{\ell}_{s,d,m} + Q_{th} \times \sum_{\mathfrak{e}_n = (\mathfrak{u}_u, \mathfrak{u}_v) \in p_{s,d} \setminus \mathfrak{u}_u \neq \mathfrak{s}_{s,d}} \mathfrak{d}_{s,d,u} \geqslant Q_{th} \times \rho^{\ell}_{s,d,m} \tag{21}$$

• In each scenario 's', the number of regenerators $\psi_{s,u,t}$ in use at node $\mathfrak{u}_u$ and time instant τ_t is equal to the sum of:

- the number of regenerators already deployed in the corresponding scenario 's' of the RRP sub-problem for the original requests that remained accepted,
- and the number of regenerators added to serve as wavelength converters and/or to cope with the QoT degradation due to the non-flat spectral response of optical equipment.

These constraints allow the WARP sub-problem to reuse, when possible, the regenerators deployed in the RRP sub-problem. $\psi_{s,u,t}$ is computed as: $\forall s = 0 \cdots N$, $\forall u = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\psi_{s,u,t} = \sum_{i=1 \cdots \widehat{D}} \widehat{\theta}_{i,t} \times \widehat{\mathfrak{d}}_{s,i,u} \times \mathfrak{a}_i + \sum_{d=1 \cdots \widetilde{D}_s} \widetilde{\theta}_{s,d,t} \times \mathfrak{d}_{s,d,u} \tag{22}$$

• The number of regenerators $\mathfrak{R}_u$ deployed at node $\mathfrak{u}_u$ is the maximum number of regenerators that are in use at any time for all the considered scenarios. $\forall s = 0 \cdots N$, $\forall u = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\mathfrak{R}_u \geqslant \psi_{s,u,t} \tag{23}$$

• A node is considered as a regeneration node if it hosts at least a single regenerator. $\forall u = 1 \cdots N$,

$$\phi_u \geqslant 10^{-3} \times \mathfrak{R}_u \tag{24}$$

• Finally, a regenerator pool failure at node $\mathfrak{u}_s$ is simulated by setting to zero the number of regenerators that can be deployed at this node in its corresponding scenario. $\forall s = 1 \cdots N$, $\forall t = 1 \cdots \mathfrak{T}$,

$$\psi_{s,s,t} = 0 \tag{25}$$

4) Objective: The objective of the WARP sub-problem remains the same as in the RRP sub-problem. We recall that this objective is expressed as:

$$\max \quad \gamma_1 \times \sum_{i=1 \cdots D} \mathfrak{a}_i - \gamma_2 \times \sum_{u=1 \cdots N} \phi_u - \gamma_3 \times \sum_{u=1 \cdots N} \mathfrak{R}_u \tag{26}$$

IV. Numerical Results

In this paper, we aim to emphasize the cost benefit brought by the $M:N$ shared regenerator protection scheme. To the best of our knowledge, this is the first paper to deal with the optimal translucent network design taking into account both

operational and backup regenerators. Thus, the only available reference scenario is the 1 : 1 regenerator protection scheme. For the sake of fairness, we consider an exact approach for the 1 : 1 regenerator protection scheme. This is achieved by recalling the work in [12] which computes the optimal number of regenerators and regeneration nodes without any consideration of network survivability. The 1 : 1 regenerator protection scheme can be derived from the latter approach by deploying two identical regenerator pools at each regeneration node; one pool of regenerators serving during normal operations and the other dedicated to backup operations.

In the sequel, we compare the results of the proposed model with the results of the reference scenario in terms of average acceptance ratio $\bar{a}$, average number of regeneration nodes $\bar{\phi}$, as well as average number of regenerators $\bar{\mathfrak{R}}$. For each request, we compute beforehand 5-shortest paths between its source and destination nodes. The parameters γ_1, γ_2, and γ_3 are set to 10^3, 1, and 10^{-3}, respectively. In other words, our main objective is to maximize the number of accepted requests, then we give higher priority to regenerator concentration over minimizing the number of regenerators. It should be noted that the number of regenerators additionally deployed in the WARP sub-problem rarely exceeds 2 regenerators. These additional regenerators are used to overcome the dependency of the signal quality on the assigned wavelength and to alleviate the wavelength continuity constraint. This demonstrates that decomposing the translucent network design problem into RRP and WARP sub-problems does not sacrifice the optimality of the final result.

A. Translucent NSF Network Design

In this section, we consider 10 sets of 200 SLDs where the activity period π is set to 0.4. For these traffic sets, we compute the optimal number of regenerators and the optimal distribution of regeneration nodes under 1 : 1 and $M : N$ regenerator protection schemes. It is worth noting that all the SLDs are accepted for all the considered sets of requests and for both protection schemes. Moreover, we should highlight that 9 nodes out of 14 (u_1, u_2, u_3, u_7, u_8, u_{11}, u_{12}, u_{13}, and u_{14}) are never selected as regeneration nodes in both protection schemes.

Under 1 : 1 regenerator protection scheme, the optimal number of regeneration nodes varies between 1 and 2 with an average value $\bar{\phi}$ equal to 1.6 regeneration nodes. The optimal number of regenerators varies between 52 and 66 with an average value $\bar{\mathfrak{R}}$ equal to 57.33 regenerators. Under $M : N$ shared regenerator protection scheme, the optimal number of regeneration nodes is always equal to 3 and the optimal number of regenerators varies between 40 and 50 with an average value $\bar{\mathfrak{R}}$ equal to 43.33 regenerators. This represents an average gain of 24.4% in terms of number of deployed regenerators.

Let us detail the solution of the translucent network design for a randomly selected set of 200 SLDs. Under 1 : 1 regenerator protection scheme, nodes u_4 and u_{10} are selected as the optimal regeneration nodes. Node u_4 contains two regenerator pools of 23 regenerators each, while node u_{10} contains two regenerator pools of 6 regenerators each. This corresponds to a total of 2 regeneration nodes and 58 regenerators. For the same set of SLDs, nodes u_4, u_5, and u_{10} are selected as the optimal regeneration nodes under $M : N$ shared regenerator protection scheme. The optimal number of regenerators at nodes u_4, u_5, and u_{10} is equal to 15, 15, and 14, respectively. This corresponds to a total of 3 regeneration nodes and 44 regenerators. During the normal operations of the network where all the regenerator pools are fully operational (Scenario '0'), 11 regenerators are used at node u_4, 12 regenerators are used at node u_5, and 6 regenerators are used at node u_{10}. When we assume that the regenerator pool at node u_4 has failed (Scenario '4'), 15 regenerators are needed at node u_5 and 14 regenerators are needed at node u_{10} in order to accommodate the 200 SLDs. Similarly, when we assume that the regenerator pool at node u_5 has failed (Scenario '5'), 15 regenerators are needed at node u_4 and 14 regenerators are needed at node u_{10}. Finally, when we assume that the regenerator pool at node u_{10} has failed (Scenario '10'), 14 regenerators are needed at node u_4 and 15 regenerators are needed at node u_5. To summarize, $N = 29$ operational regenerators are protected using $M = 44 - 29 = 15$ backup regenerators. This highlights the cost benefit brought by the $M : N$ shared regenerator protection scheme.

Furthermore, it should be noted that the number of SLDs that required regeneration varies between 40 and 41 requests according to the considered failure scenario. If we use a dedicated regenerator for each SLD requiring regeneration, the resulting routing solution would require 40 or 41 regenerators. However, thanks to the resource reutilization between time-disjoint SLDs, the number of regenerators used in these scenarios is only equal to 29 regenerators.

To conclude, we note that the optimal number of regeneration nodes for the NSF network under medium traffic loads, while ensuring network survivability, is equal to 3 nodes. Concentrating regenerators into 1 or 2 nodes exposes the network to a high risk of data loss in the eventual case of regenerator pool failure. Furthermore, we note that $M : N$ shared regenerator protection scheme evenly distributes the number of required regenerators over the regeneration nodes.

B. Impact of Traffic Load on Regenerator Distribution

In this section, we consider three different loads of permanent requests ($D \in \{100, 200, 300\}$). For each traffic load, we generate 10 sets of PLDs. Table II summarizes the results obtained for the different traffic loads considered in our evaluation. Figure 3 shows the *median* distribution of the deployed regenerators over the network nodes. It is obvious that the number of regenerators and regeneration nodes increase with the traffic load. For 100 PLDs, the $M : N$ and 1 : 1 protection schemes achieve the same results. However, the $M : N$ shared regenerator protection scheme achieves in average a reduction of 22% to 25% in the number of deployed regenerators compared to the 1 : 1 regenerator protection scheme for the sets of 200 and 300 PLDs.

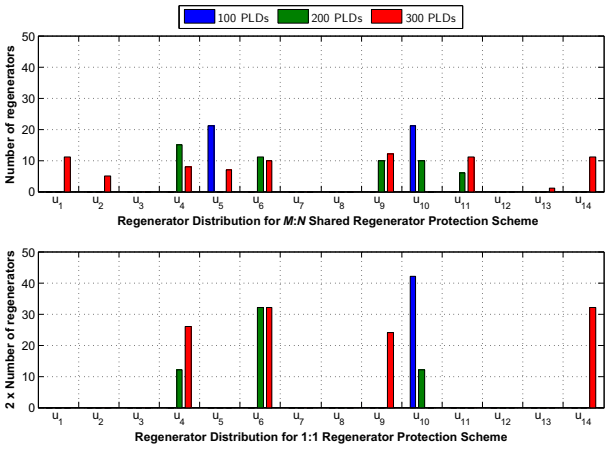


Fig. 3. Median regenerator distribution $\mathfrak{R}_u$ for various loads of PLDs.

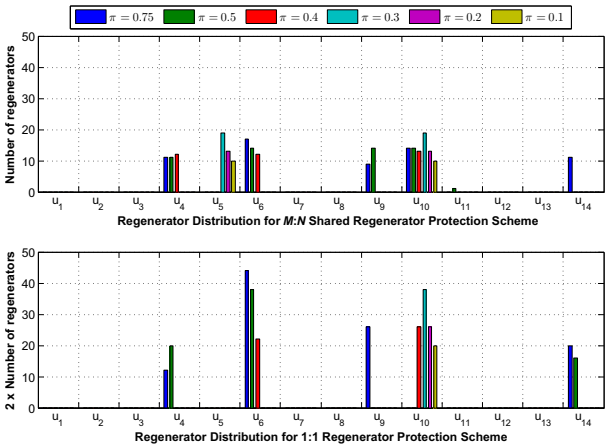


Fig. 4. Median regenerator distribution $\mathfrak{R}_u$ for various sets of 200 SLDs.

TABLE II
RESULTS FOR VARIOUS LOADS OF PERMANENT PLDS.

D	$M:N$ protection			1 : 1 protection		
	$\bar{\mathfrak{a}}$	$\bar{\phi}$	$\bar{\mathfrak{R}}$	$\bar{\mathfrak{a}}$	$\bar{\phi}$	$\bar{\mathfrak{R}}$
100	100%	2	42	100%	1	42
200	100%	4.33	48.67	100%	2.67	62.67
300	87.56%	9.33	87.33	88.33%	4	119.33

TABLE III
RESULTS FOR VARIOUS SETS OF 200 DYNAMIC SLDS.

π	$M:N$ protection			1 : 1 protection		
	$\bar{\mathfrak{a}}$	$\bar{\phi}$	$\bar{\mathfrak{R}}$	$\bar{\mathfrak{a}}$	$\bar{\phi}$	$\bar{\mathfrak{R}}$
0.75	100%	5.67	68	100%	3	96.67
0.5	100%	5.33	65.33	100%	3	85.33
0.4	100%	3	43.33	100%	1.6	57.33
0.3	100%	2	40.67	100%	1	40.67
0.2	100%	2	28.67	100%	1	28.67
0.1	100%	2	18.67	100%	1	18.67

C. Impact of Time-Correlation on Regenerator Distribution

In this section, we investigate the impact of the requests' time-correlation on the number of regenerators and regeneration nodes by considering dynamic requests with different activity periods ($\pi \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.75\}$). For each value of the time-correlation, we generate 10 sets of 200 SLDs. Table III summarizes the results obtained for the different sets of SLDs. Figure 4 shows the *median* distribution of the deployed regenerators over the network nodes. We notice that for small values of π ($\pi \in \{0.1, 0.2, 0.3\}$), the $M:N$ and $1:1$ regenerator protection schemes achieve the same results, and the nodes u_5 and u_{10} are the only regeneration nodes. For large values of π ($\pi \in \{0.4, 0.5, 0.75\}$), nodes u_4, u_6, u_9, and u_{10} host more than 80% of the deployed regenerators. Moreover, for the latter values, the reduction in the number of deployed regenerators varies between 23% and 30% when comparing the $M:N$ and $1:1$ regenerator protection schemes.

D. Translucent EBN Network Design

In this section, we consider the EBN backbone network and generate 10 different sets of 150 SLD requests with an activity period π of 0.4. In the case of $1:1$ protection scheme, all the SLDs are accepted, while in the $M:N$ protection scheme, the number of rejected SLDs varies between 10 and 19 with an average value of 14 rejected requests. This corresponds to an average acceptance ratio $\bar{\mathfrak{a}}$ of 90.66%. It is worth noting that nodes u_1, u_2, u_4, u_8, u_{13}, u_{18}, u_{20}, and u_{21} are never selected as regeneration nodes in both protection schemes.

In the case of $1:1$ protection scheme, the optimal number of regeneration nodes is always equal to 4 and the optimal number of regenerators varies between 96 and 132 with an average value $\bar{\mathfrak{R}}$ equal to 108 regenerators. In the case of $M:N$ protection scheme, the optimal number of regeneration nodes varies between 4 and 6 with an average value $\bar{\phi}$ equal to 5.6 regeneration nodes, while the optimal number of regenerators varies between 41 and 66 with an average value $\bar{\mathfrak{R}}$ equal to 51.4 regenerators. However, it is not fair to compute the average gain as the two protection schemes do not have the same rejection ratio.

V. CONCLUSION

Reducing the number of regenerators and regeneration nodes is highly motivated by the reduction in power consumption and maintenance cost. However, excessively concentrating the regenerators into a small number of nodes exposes the network to a high risk of data loss in the hazardous event of a regenerator pool failure. The same analysis is applicable if we investigate the failure of multi-channel all-optical regenerators instead of electrical regenerators pool. Indeed, a multi-channel all-optical regenerator can be used to simultaneously regenerate several wavelengths and the failure of such device will impact all the requests that are planned to be regenerated at this node.

Therefore, it is essential to keep in mind the network survivability concern while dimensioning the network. In this paper, we propose an exact approach based on a mathematical formulation that implements an $M:N$ shared regenerator protection scheme where N operational regenerators are protected using M backup regenerators. The proposed formulation compute the optimal number of regeneration nodes and seeks to evenly distribute the number of required regenerators over the regeneration nodes. In order to improve the scalability of our approach, we decompose the problem into the "Routing and

Regenerator Placement" (RRP) sub-problem and the "Wavelength Assignment and Regenerator Placement" (WARP) sub-problem. We showed that decomposing the original problem into two sub-problems does not sacrifice the optimality of the final result. Furthermore, in order to shorten the time needed to reach the optimal regenerator deployment solution, we tighten the formulation by adding constraints that will help discard equivalent solutions without omitting the optimal solution.

As a rule of thumb, we can conclude that when the deployed regenerators can be concentrated into a single regeneration node under the $1:1$ regenerator protection scheme, the $M:N$ shared regenerator protection scheme achieves comparable results to those obtained by the $1:1$ regenerator protection scheme. This is usually achieved by equally splitting the number of required regenerators over two distinct regeneration nodes. However, when the number of regenerator nodes increases, the $M:N$ shared regenerator protection scheme outperforms its counterparts by evenly distributing the number of required regenerators over several regeneration nodes. The gain obtained by the $M:N$ shared regenerator protection scheme may rapidly exceed 25% in terms of number of deployed regenerators.

REFERENCES

[1] T. Schmidt, C. Malouin, R. Saunders, J. Hong, and R. Marcoccia, "Mitigating channel impairments in high capacity serial 40 G and 100 G DWDM transmission systems," in *Digest of the IEEE/LEOS Summer Topical Meetings*, 2008, pp. 141–142.

[2] A. Morea, N. Brogard, F. Leplingard, J.-C. Antona, T. Zami, B. Lavigne, and D. Bayart, "QoT function and a* routing: an optimized combination for connection search in translucent networks," *OSA JON*, vol. 7, no. 1, pp. 42–61, Jan. 2008.

[3] X. Yang and B. Ramamurthy, "Sparse regeneration in translucent wavelength-routed optical networks: architecture, network design and wavelength routing," *Springer PNC*, vol. 10, no. 1, pp. 39–50, Jul. 2005.

[4] G. Shen, W. Grover, T. Cheng, and S. Bose, "Sparse placement of electronic switching nodes for low-blocking in translucent optical networks," *OSA JON*, vol. 1, no. 12, pp. 424–441, Dec. 2002.

[5] Q. Rahman, Y. Aneja, S. Bandyopadhyay, and A. Jaekel, "Optimal regenerator placement in survivable translucent networks," in *Proc. of IEEE DRCN*, 2014.

[6] R. Ramlall, Q. Rahman, Y. Aneja, and S. Bandyopadhyay, "Optimal regenerator placement for path protection in impairment-aware WDM networks," in *Proc. of IEEE HPSR*, 2015.

[7] M. Youssef, S. Al Zahr, and M. Gagnaire, "Translucent network design from a CapEx/OpEx perspective," *Springer PNC*, vol. 22, no. 1, pp. 85–97, Aug. 2011.

[8] S. Pachnicke, T. Paschenda, and P. Krummrich, "Assessment of a constraint-based routing algorithm for translucent 10 Gbits/s DWDM networks considering fiber nonlinearities," *OSA JON*, vol. 7, no. 4, pp. 365–377, Apr. 2008.

[9] Z. Pan, B. Chatelain, D. Plant, F. Gagnon, C. Tremblay, and E. Bernier, "Tabu search optimization in translucent network regenerator allocation," in *Proc. of IEEE BROADNETS*, 2008, pp. 627–631.

[10] W. Zhang, J. Tang, K. Nygard, and C. Wang, "Repare: Regenerator placement and routing establishment in translucent networks," in *Proc. of IEEE GLOBECOM*, 2009, pp. 1–7.

[11] K. Manousakis, K. Christodoulopoulos, E. Kamitsas, I. Tomkos, and E. Varvarigos, "Offline impairment-aware routing and wavelength assignment algorithms in translucent WDM optical networks," *IEEE/OSA JLT*, vol. 27, no. 12, pp. 1866–1877, June 2009.

[12] E. A. Doumith, S. Al Zahr, and M. Gagnaire, "Mutual impact of traffic correlation and regenerator concentration in translucent WDM networks," in *Proc. of IEEE ICC*, 2011, pp. 1–6.

[13] S. Al Zahr, E. A. Doumith, and M. Gagnaire, "An exact approach for translucent WDM network design considering scheduled lightpath demands," in *Proc. of IEEE ICT*, 2011, pp. 450–457.

[14] M. Gagnaire, E. A. Doumith, and S. Al Zahr, "A novel exact approach for translucent WDM network design under traffic uncertainty," in *Proc. of IEEE ONDM*, 2011, pp. 1–6.

[15] F. Parmigiani, L. Provost, P. Petropoulos, D. J. Richardson, W. Freude, J. Leuthold, A. D. Ellis, and I. Tomkos, "Progress in multichannel all-optical regeneration based on fiber technology," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 18, no. 2, pp. 689–700, March 2012.

[16] S. Gringeri, V. Basch, B.and Shukla, R. Egorov, and T. J. Xia, "Flexible architectures for optical transport nodes and networks," *IEEE Communications Magazine*, vol. 48, no. 7, pp. 40–50, Jul. 2010.

[17] G. P. Agrawal, *Fiber optic communication systems*. Wiley-Interscience Publication, 1997.

[18] S. Al Zahr, "WDM translucent networks planning with guaranteed quality of transmission," https://pastel.archives-ouvertes.fr/pastel-00004014, École Nationale Supérieure des Télécommunications, Paris - France, 2007.

SRv6Pipes: enabling in-network bytestream functions

Fabien Duchene, David Lebrun, Olivier Bonaventure
ICTEAM, Université catholique de Louvain
Louvain-la-Neuve, Belgium
Email: firstname.lastname@uclouvain.be

***Abstract*—IPv6 Segment Routing is a recent IPv6 extension that is generating a lot of interest among researchers and in industry. Thanks to IPv6 SR, network operators can better control the paths followed by packets inside their networks. This provides enhanced traffic engineering capabilities and is key to support Service Function Chaining (SFC). With SFC, an end-to-end service is the composition of a series of in-network services. Simple services such as NAT, accounting or stateless firewalls can be implemented on a per-packet basis. However, more interesting services like transparent proxies, transparent compression or encryption, transcoding, etc. require functions that operate on the bytestream.**

In this paper, we extend the IPv6 implementation of Segment Routing in the Linux kernel to enable network functions that operate on the bytestream and not on a per-packet basis. Our SRv6Pipes enable network architects to design end-to-end services as a series of in-network functions. We evaluate the performance of our implementation with different microbenchmarks.

I. Introduction

Middleboxes play an important role in today's enterprise and datacenter networks. In addition to the traditional switches and routers, enterprise networks contain other devices that forward, inspect, modify or control packets. There is a wide variety of middleboxes [1], ranging from simple NAT, IP firewalls, various forms of Deep Packet Inspection, TCP Performance Enhancing Proxies (PEP), load balancers, Application Level Gateways (ALG), proxies, caches, edge servers, etc. Measurement studies have shown that some networks have deployed as many middleboxes as the number of traditional routers [2].

Those middleboxes were not part of the original TCP/IP architecture. They are typically deployed by either placing the middleboxes on the path of the traffic that needs to be handled, *e.g.*, on the link between two adjacent routers, or by using specific routing configurations to force some packets to pass through a particular middlebox. These two deployment approaches are fragile and can cause failures that are hard to diagnose and correct in large networks. Pothraju and Jain have shown in [3] that middlebox failures are significant and that many of them belong to a grey zone, *i.e.*, they cause link flapping or connectivity errors that are difficult to debug and impact the end-to-end traffic. Researchers and vendors have proposed Network Function Virtualization (NFV) [4] and Service Function Chaining (SFC) [5] to solve some of the problems caused by middleboxes.

ISBN 978-3-903176-08-9

In a nutshell, the NFV paradigm argues that all network functions should be virtualised and executed on commodity hardware instead of requiring specific devices. On the other hand, SFC [5] proposes to support chains of network functions which can be applied to the packets exchanged between communicating hosts. Several realisations for SFC are being discussed within the IETF. The SFC working group is developing the Network Service Header [6]. This new header can be used to implement service chains and replaces already deployed proprietary solutions. Another approach is to leverage the extensibility of IPv6. Given the global deployment of IPv6 [7], several large enterprises have already announced plans to migrate their internal network or their datacenters to IPv6-only to avoid the burden of managing two different networking stacks [8]. In addition to having a larger addressing space than IPv4, IPv6 provides several interesting features to support middleboxes in enterprise and datacenter networks. One of these is the native support for Segment Routing [9], [10]. Segment Routing (SR) is a modern variant of source routing that enables network administrators to enforce specific network paths.

In this paper, we demonstrate the benefits that the IPv6 Segment Routing (SRv6) architecture can bring to support middleboxes in enterprise and datacenter networks. With SRv6, middleboxes can be exposed in the architecture and visible end-to-end. This significantly improves the manageability of the network and the detection of failures while enabling new use cases where applications can select to use specific middleboxes for some end-to-end flows. This paper is organized as follows.

In Section II, we describe some use cases that can benefit from middleboxes. In Section III, we present *SRv6Pipes*, a modular SRv6-based architecture to support arbitrary in-network Virtual Functions, that can be applied on bytestreams and chained together. In Section IV, we detail a prototype implementation of our architecture, running on Linux. In Section V, we demonstrate the feasibility of our approach and evaluate the performance of our prototype through various tests and microbenchmarks. Finally, we cover some related work in Section VI and conclude in Section VII. Future work is discussed in VIII.

II. Use Cases

Middleboxes can perform two different types of network functions: *per-packet* and *per-bytestream*. The *per-packet* functions operate on a per-packet basis. They include Network Address Translation and simple firewalls. These functions typically operate on the network and sometimes transport headers. The *per-bytestream* functions are more complex, but also more useful. These functions operate on the payload of the TCP packets. For example, firewalls and Intrusion Detection Systems (IDS) need to match patterns in the packet payload while transparent compression and/or encryption need to modify the payload of TCP packets. Such functions need to at least reorder the received TCP packets but often need to include an almost complete TCP implementation. We describe some of these *per-bytestream* functions in more details in this section.

A. Application-level Firewalling

To cope with various forms of packet reordering, application-level firewalls and Intrusion Detection/Prevention Systems need to at least normalize the received packets [11] before processing them. Another approach is to use a transparent TCP proxy on the firewall to terminate the TCP connection and let the firewall/IDS process the reassembled payload. An end-to-end connection would thus be composed of two sub-connections: one between the client and the middlebox and another one between the middlebox and the server.

Network operators often configure access lists to associate IP prefixes to some security checks performed by the IDS. For example, students would be subject to different policies than servers.

B. Multipath TCP Proxies

Multipath TCP [12] (MPTCP) is a recent TCP extension that enables hosts to send packets belonging to one connection over different paths. One of the benefits of MPTCP is that it allows to aggregate the bandwidth of multiple connections. This enables, *e.g.*, network operators to bond xDSL and LTE networks to better serve rural areas [13]. However, MPTCP being an end-to-end protocol, the client and the server require an MPTCP-enabled kernels. To leverage the benefits of MPTCP without modifying the client or server network stacks, operators started developing MPTCP-aware proxies [13], [14] to convert regular TCP to MPTCP and conversely.

To allow the bundling of xDSL and LTE, an NFV deployment could be leveraged to implement the same behavior, by placing a proxy in the CPE to convert regular TCP to MPTCP and a second proxy in the operator's network to convert MPTCP to regular TCP. This would allow non-MPTCP clients and servers to use different networks simultaneously.

In practice, network operators could want to support different services on the same proxy, e.g. *(i)* a business proxy that always maximizes bandwidth for business customers, *(ii)* a low-cost proxy that only uses the LTE network when the xDSL network is fully utilized or *(iii)* a gaming proxy that always uses the network that provides the lowest delay. Such proxies can be deployed by tuning the packet scheduler and the path manager of Multipath TCP implementations.

C. Multimedia transcoding

Multimedia transcoding has been a research topic for a long time [15], [16]. Since, it has been widely deployed by companies like Amazon [17]. In this context, a proxy placed between the client and the server that hosts the multimedia file can be used to transcode the multimedia file hosted on the server into a format compatible with the client. This allows to distribute the computation intensive task of transcoding the content over several proxies, while the server simply hosts the original files. In this setup, parameters could be passed to the proxy to specify for instance the maximum bitrate that a client is entitled to (based on technical or subscription limitations), the maximum number of streams allowed for this client or the type of content authorized for this client.

III. Architecture

Middleboxes and other in-network functions are installed, configured, and managed by network administrators according to business (e.g. security regulations impose the utilisation firewalls) and technical (e.g. performance issues force the utilisation of performance enhancing proxies, or addressing issues force the utilisation of NAT) needs. Usually, network administrators impose the utilisation of specific network functions by configuring routing policies or placing physical boxes on links that carry specific traffic (e.g. firewalls are often attached to egress links). This is both cumbersome and costly since all possible links must be covered by each intended network function.

Like NFV, our architecture assumes that network functions are software modules which can be executed anywhere in the network. A firewall function that only needs to process the external flows does not need to be installed on the egress router, it can be executed on any server or router inside an enterprise network. Each network function is identified by an IPv6 prefix which is advertised by the equipment hosting the function (see section III-C). For redundancy or load-balancing, the same function can be hosted on different equipments in the network.

To understand the different elements of our architecture, let us consider a simple scenario. A client host needs to open a TCP connection towards a remote server. The network administrator has decided that the packets belonging to such a connection must be processed by two network functions: *(i)* a stateless firewall which blocks prohibited ports and *(ii)* a DPI which inspects all external TCP connections. Three elements of our architecture are used to support this sequence of network functions in enterprise networks.

The first element is IPv6 Segment Routing (SRv6) [18]. Our architecture uses the SRv6 header (SRH) to enforce an end-to-end path between the client and the server which passes through the two equipments hosting the mandatory networking functions. We describe SRv6 in more details in section III-A.

The second element of our architecture is how the client learns the SRH suitable to reach a given destination. For this, we modify the enterprise DNS resolver. Instead of simply resolving names into addresses, our DNS resolver acts as a controller [19], [20] which has been configured by the network administrator with various network policies. When a client sends a DNS request to the resolver, it replies with the intended response and additional records which contain the SRH that the client has to apply to reach the specified addresses.

Thanks to the SRH which is attached by the client, all the packets belonging to the TCP connection will pass through the stateless firewall and the DPI. Consider now what happens if some packets are lost and need to be retransmitted. The stateless firewall is not affected since it only processes the network and transport headers that are present in each packet. On the other hand, the DPI function needs to include a TCP implementation to be able to detect out-of-order packets or other TCP artifacts. Instead of requiring each network function to include a TCP implementation, our architecture leverages the TCP stack that is already present in the Linux kernel. Each equipment that hosts a network function uses a transparent TCP proxy that transparently terminates the TCP connections and exposes bytestreams to the network functions as in FlowOS [21]. This greatly simplifies the implementation of per-bytestream network functions

A. IPv6 Segment Routing

Segment Routing (SR) is a modern variant of the source routing paradigm, currently under standardization at the IETF [18]. SR can be used on top of an MPLS or IPv6 dataplane to steer packets through an ordered list of *segments*. SR is now well-supported on commercial routers [23] and Linux hosts [24] and deployed by major ISPs [9].

The IPv6 flavor of Segment Routing (SRv6) leverages a dedicated IPv6 routing extension header, named Segment Routing Header (SRH) [10]. Each segment is an IPv6 address representing a node or link to traverse, or an intermediate function to be executed. The SRH contains a full list of segments. The active segment is referenced by an index, the segment pointer. As the list of segments is encoded in reverse order, the index is first initialized to the last element of the list (*i.e.*, the first segment of the path), and decremented at each *segment endpoint*. The segment pointer thus reaches zero when arriving at the last segment of the path. The active segment is also written as the destination address in the IPv6 header. As such, transit nodes on the path to an active segment simply needs to support plain IPv6 forwarding. SRv6 support is only required at the segment endpoints.

In SRv6Pipes, we leverage the SRv6 architecture to steer TCP flows through arbitrary network functions. See Figure 1 for an illustration. Consider that the client `C` establishes a connection to a server `S`, with two intermediate network functions at `P1` and `P2`. To realise that, the client attaches an SRH to its packets, containing three segments. The first two segments represent the functions to be executed at resp. `P1` and `P2`. The third segment is the address of `S`. When the packets are transiting between `C` and `P1`, and between `P1` and `P2`, their IPv6 destination address is thus the address of the function to execute at the corresponding proxy. Between `P2` and `S`, the segment pointer is decremented to zero and the IPv6 destination address of the packets is the address of `S`.

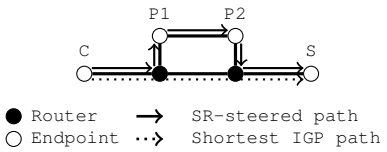


Fig. 1: Traffic steering through two off-path network functions `P1` and `P2` (*e.g.*, firewall and IDS).

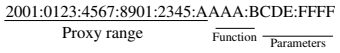


Fig. 2: IPv6 address encoding.

B. Transparent TCP Proxy

The proxy is the core component of our architecture to support per-bytestream network functions. It is transparent at the network layer, meaning that even if the proxy actually terminates the TCP connection with the client, the destination server will receive packets coming from the client's IP address, and not from the proxy's IP address. The transparent proxy is placed on path using the IPv6 Segment Routing Header (SRH) [10]. It intercepts each new connection that matches a given pattern (*e.g.*, a destination port) and terminates it. Then, the proxy establishes a downstream connection to the next segment specified in the SRH of the inbound connection. When the proxy receives data from the client, it applies a transformation function (*i.e.*, the Virtual Function) to the received data and forwards the result on its outbound connection to the next segment of the path. This process is then repeated until reaching the final destination of the path.

C. Encoding Functions and Parameters

As shown in section II, some parameters can be passed to the per-bytestream function. Such parameters are usually specified in the proxy configuration files. However, such configurations can be large and complex if some parameters can change on a per connection basis. Consider for example a first proxy that encrypts the payload and a second that decrypts it. Those encryption/decryption proxies would have to be configured with the encryption/decryption keys for each flow. A possible approach would be to define one key per host or set of hosts. A better approach is to configure a set of keys on the proxies and associate each key with a unique identifier. When a connection starts, the encryption proxy selects a random key and places the identifier of the chosen key in the SRH towards the decryption proxy.

To enable such a granularity in the choice of transformation functions and parameters, we leverage the large addressing space available in IPv6. Each proxy announces one or more IPv6 prefixes that correspond to the Virtual Functions it hosts.

Within the host part of the prefixes, we allocate a given amount of bits to encode the identifier of the function to apply as proposed in [25]. The remaining low order bits are used to specify parameters of the virtual function such as the decryption key in the above example. Consider Figure 2 for an illustration. The proxies announce `/80` prefixes. The first 80 bits of the address thus specify the proxy to traverse. The 16 following bits identify the function to apply to the payload, and the low-order 32 bits contain the parameters of these functions. The SRH then contains a list of proxies with their respective functions and parameters. This approach allows the clients to use any combination of function/parameter available in the network.

Consider the network described in figure 1. In this network, the client might require to encrypt the traffic between `P1` and `P2`. In our architecture, the client will use the `function` bits of the address of `P1` to specify the identifier of the `encrypt` function, and the `parameters` bits to specify the identifier of an encryption key. The same will be done in the address of `P2` with the `decrypt` function. This allows to have different encryption keys for different connections without having to store a configuration for each connection in the proxy. The processing of the return traffic is discussed in IV-E.

D. SRv6 Controller

In our architecture, a TCP client is able to specify arbitrary functions to apply to its traffic. However, keeping track of all the functions, parameters, and proxies addresses represents a significant amount of complexity. This complexity can be abstracted by a central SDN-like controller. We leverage the *SDN Resolver*, which is a DNS-based, SRv6 controller introduced in [19], [20]. Before establishing a connection, the client sends a request to the controller with the address of the server and a list of functions to apply to the traffic. The controller then computes a path that matches the request and returns an SRH to the client. A key element of this controller is that the SRH returned to the client does not contain the full list of segments. Instead, it contains only one segment, called the *binding segment*. The access router of the client is configured by the controller to translate this binding segment into the full list of segments. This abstraction enables the clients to be oblivious to changes in the SRH induced by, *e.g.*, a network failure. The architectural and implementation details of *SDN Resolver* are available in [19], [20]. Note that the DNS protocol serves as an example, that can be replaced by any ad-hoc application-facing protocol.

E. Security Considerations

The ability to execute and chain arbitrary functions in the network has obvious security implications. To restrict the privilege of using SRv6Pipes proxies, we can leverage the central controller presented in the previous section, as well as its *binding segment* mechanism. By configuring all access routers to accept only SRHs with known binding segments, we can effectively prevent an uncontrolled usage of network functions. The decision to accept or deny the use of a given set of functions is made by the controller, which can identify clients through independent channels [19].

IV. Implementation

To demonstrate the feasibility of our approach, we implemented a prototype of our solution by extending the implementation of IPv6 Segment Routing in the Linux kernel [24]. The main new component of our prototype is a transparent, SR-aware, TCP proxy. For this, we extended the kernel implementation of SRv6 with a new type of function. An overview of the various data paths in our prototype is shown in Figure 3.

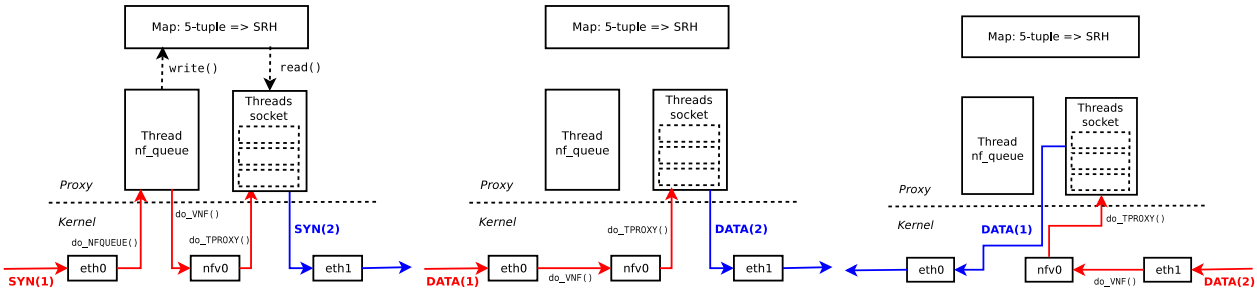
To ensure that our proof of concept could easily be used to reproduce our results on any off-the-shelf hardware, we implemented it using the regular Linux mechanisms. Alternatives solutions are discussed in Section VIII

A. Transparent SR-Aware TCP Proxy

The core objective of our proxy is to process and relay TCP streams between two segments of a segment routed path. To achieve this, the proxy must (i) intercept and terminate incoming TCP flows, (ii) optionally apply transformation functions to the bytestreams, and (iii) initiate and maintain the corresponding TCP flows to the next segment of the path.

To intercept TCP flows, the proxy must accept connections towards pairs of IP/port that are not local to the machine, which is not possible by default. The Linux kernel provides the `TPROXY` iptables extensions, enabling such interceptions. It works by redirecting all packets matching an iptables rule towards a local IP/port pair. The proxy is then able to intercept the corresponding TCP flows by listening to this local pair.

Once a TCP flow is intercepted and terminated, the proxy needs to retrieve the associated SRH, decrement its segment pointer, and install it on the corresponding outbound socket. The `IPV6_RECVRTHDR` socket option could be used to fetch any attached Routing Header (RH) as ancillary data, using the `recvmsg()` system call. However, this feature is only implemented for datagram protocols such as UDP, where a single RH is associated to each datagram. In bytestream protocols such as TCP, packets can be merged and the $1:1$ mapping to RHs is lost. In our prototype, we rely on the SRH included in the SYN packet of a given TCP flow. As the kernel does not expose Routing Headers for TCP flows, we leverage the `NFQUEUE` iptables extension to capture SYN packets in user space. The proxy opens a netlink channel with the kernel and receives through it all SYN packets matching the corresponding iptables rule. Then, the proxy extracts the 5-tuple and the SRH from the SYN packet and stores them in a `flows_srh` map. Finally, the packet is reinjected into the kernel. Following its normal data path, the SYN packet will trigger a connection request to the proxy. Using the 5-tuple, the proxy is then able to retrieve the SRH previously stored in the `flows_srh` map. While capturing every packet in user space can severely degrade the performances, our solution does not suffer from such degradation as we only capture the first packet of each flow.



(a) Traversal of a SYN packet through the proxy. The SRH is recorded for the 5-tuple.

(b) Traversal of data packets.

(c) Traversal of return packets.

Fig. 3: Overview of possible data paths within SRv6Pipes.

After having intercepted a TCP flow and extracted its SRH, the proxy must establish the corresponding TCP flow to the next iteration of the path. To achieve this, the proxy creates the outbound socket and attaches to it the corresponding SRH. Additionally, the connection must appear as originating from the actual source of the flow. Using the `IP_FREEBIND` socket option, the proxy is able to bind to a non-local IP/port pair. Finally, the connection is established and data can be exchanged.

Once both connections (inbound and outbound) are established, the proxy only needs to forward data coming from one socket to the other one, after going through an optional transformation function. In our prototype, we use an application-level buffer to transfer data from one connection to the other. Another possible solution would be to use the `splice()` system call to let the kernel directly move data between file descriptors. However, this solution prevents the proxy from actually modifying the data. Our approach allows the implementation of arbitrary transformation functions. The termination of connections is straightforward. Once one socket is closed, any in-flight data is flushed and the other socket is also closed.

We implemented a multi-threaded architecture, enabling the proxy to scale with the load. One dedicated thread handles the `NFQUEUE` channel, receives the SYN packets, and populates the `flows_srh` map accordingly. A configurable number of threads (typically one per CPU thread) accept incoming connections, establish the outbound connection, and process the data exchanged between them. Each of these threads leverages the `SO_REUSEPORT` socket option, enabling them to simultaneously listen to the same local IP/port. The result is that the kernel maintains distinct accept queues for each thread. Consequently, incoming connections are equally load-balanced across the running threads.

B. Kernel Extensions

When a packet to be processed by the proxy enters the kernel, its IPv6 destination address corresponds to the local proxy function. However, the TCP checksum was originally computed for the actual destination of the packet. As such, it is transiently incorrect, due to the SR-triggered change of destination address. Additionally, the packet will be associated to the proxy's local socket by the `TPROXY` module, and subsequently injected in the local stack. However, the segment pointer of the associated SRH is non-zero. The packet will thus enter the SRH processing and the kernel will attempt to forward it to the next segment, bypassing the local TCP processing [24].

To address those two issues, we extend the SRv6 kernel implementation available in Linux 4.14 and add a new type of function called `End.VNF`. This function takes one parameter (an egress interface) and performs the following actions. First, it updates the destination address of the packet to its final destination. Then, it sets the segment pointer to zero[1]. Finally, it injects the resulting packet into the specified egress interface using `netif_rx()`. In our prototype, we leverage a virtual dummy interface (`nfv0`). As a result, all packets to be intercepted by the proxy are received through this particular interface and are thus easily distinguished from background traffic.

C. System Configuration

To instantiate the proxy, a non-trivial configuration of iptables and routing tables is required. An example of this configuration is shown in Figure 4. The first two lines create the `nfv0` interface to receive all packets to be intercepted by the proxy. Lines $3-5$ create a `DIVERT` iptable chain that sets the mark `1` on packets and accepts them. Line 6 creates an `NFQUEUE` rule that matches all SYN packets whose destination address corresponds to the local proxy (`PROXY_FUNC_ADDR`) and sends them to the queue number 0. Line 7 matches all TCP packets received on interface `nfv0` and sends them to the `TPROXY` target. The latter will set the mark `1` on those packets and will associate them to a socket bound on a local `PROXY_LOCAL_PORT` port. Line 8 matches all TCP packets that can be associated to an open socket and sends them to the previously configured `DIVERT` chain. In practice, this rule will catch the inbound return packets that are not caught by the two previous rules. Line

[1]As the SRH of the SYN packet was previously extracted by the proxy, this information is not lost.

```
1: ip link add nfv0 type dummy
2: ifconfig nfv0 up
3: ip6tables -t mangle -N DIVERT
4: ip6tables -t mangle -A DIVERT -j MARK --set-mark 1
5: ip6tables -t mangle -A DIVERT -j ACCEPT
6: ip6tables -t mangle -A PREROUTING -d \$PROXY_FUNC_ADDR -p tcp --syn -j NFQUEUE --queue-num 0
7: ip6tables -t mangle -A PREROUTING -i nfv0 -p tcp -j TPROXY --tproxy-mark 0x1/0x1 --on-port \$PROXY_LOCAL_PORT
8: ip6tables -t mangle -A PREROUTING -p tcp -m socket -j DIVERT
9: ip -6 rule add fwmark 1 table 100
10: ip -6 route add local ::/0 dev lo table 100
11: ip -6 route add \$PROXY_FUNC_ADDR/128 encap seg6local action End.VNF oif nfv0 dev eth0
12: sysctl net.ipv6.conf.nfv0.seg6_enabled=1
```

Fig. 4: System configuration for the proxy.

9 creates a routing rule instructing the kernel to lookup table 100 for all packets having the mark `1`. Line 10 creates a single routing entry into table 100 that matches all packets and sends them in the local stack (instead of forwarding them). Line 11 creates an SRv6 routing entry that matches all packets towards `PROXY_FUNC_ADDR` and applies the `End.VNF` function, using `nfv0` as the egress interface[2]. Finally, line 12 enables the processing of SRv6 packets on interface `nfv0`.

D. Modular Transformation Functions

To support transformation functions in a modular way, our SRv6Pipes proxy leverages Linux dynamic libraries. Functions can be compiled in `.so` (shared object) files. Those files are independent modules that can be loaded and unloaded at run-time by the proxy. Each module exports an `all_funcs` symbol. This symbol refers to an array of `func_desc` structures. Each of those structures describes a single transformation function, through the following symbols. The `func_init()` symbol is called once, on module load. It registers the function with a given function identifier, which is passed in the IPv6 destination addresses (see Section III-C). The `func_spawn()` symbol is called each time a new intercepted TCP flow matches the function identifier. Any parameter passed in the low-order bits of the IPv6 destination address is passed as argument. The role of this symbol is to initialize per-connection data. The `func_process()` symbol is the actual transformation function. It reads data from an input buffer and writes the transformed data in an output buffer. The `func_despawn()` symbol is called at connection termination and it frees previously allocated per-connection data. Finally, the `func_deinit()` symbol is called at module unload and de-registers function identifiers.

Such an architecture enables to easily add, modify, and remove transformation functions, without updating nor restarting the proxy's binary.

E. Return Traffic

The previous sections detailed the processing of the upstream traffic (from client to server). However, if the middleboxes are not located in-path, the downstream traffic (from the server to client) must also be augmented with an SRH. This is also necessary to enable asymmetrical processing functions, *i.e.*, using different transformation functions depending on the direction of the traffic. To achieve this, multiple options exist.

The straightforward option is to simply "*reverse*" the SRH received from the client or from the previous proxy. Each proxy can simply apply the segments of the initial SRH in reverse order. While this solution is simple and does not incur a significant overhead, it as a major limitation: the segments must necessarily be symmetrical, making asymmetrical processing functions impossible.

To enable asymmetrical processing functions, another option is to embed the return SRH in a TLV extension of the initial SRH. With this solution, after inserting the SRH, the client inserts a TLV to the socket before establishing the connection. Then, each proxy and the server extract the SRH to be used on the return path from the TLV received in the initial packet (SYN). The TLV could also be transmitted with every upstream packet, but this would increase the overhead. With this TLV, it is important to note that the return path must include every proxy that is present in the upstream path, but that others segments, e.g. corresponding to specific paths or routers, can be added or suppressed.

In our prototype, we implemented the second solution by modifying the Linux kernel to add support for such a TLV. When a new TCP socket is created after receiving an SR-enabled SYN packet containing the return-path TLV, this return path is extracted and installed as an outbound SRH for the newly created socket. If the proxies are located in-path, our prototype can also work without an SRH on the return path. This is realized using the `DIVERT` rules shown in figure 4. In Section V, we evaluate this on-path mode.

V. Evaluation

In this section, we use microbenchmarks to evaluate the performance of our prototype in our lab. For this evaluation, we use three Linux PCs connected with 10Gbps interfaces as shown in figure 5.

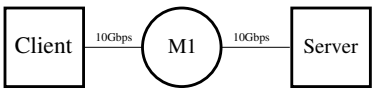


Fig. 5: Lab setup. M1 can be configured as router or proxy.

[2]While this interface is considered egress from the point of view of `End.VNF`, packets are actually received on that interface and it is thus considered ingress for the next components in the datapath.

The client is a 2,53Ghz Intel Xeon X3440 with 16GB of RAM. M1 and the server use the same hardware configuration but with only 8GB of RAM. They are all equipped with Intel 82599 10Gbps Ethernet adapters and use 9000 bytes MTU. They all use our modified version of the latest IPv6 Segment Routing kernel based on the Linux kernel version 4.14. The server runs `lighttpd` version 1.4.35. The client uses `wrk` [26] 4.0.2-5 to load the server with HTTP 1.1 requests. We slightly modified `wrk` to add an IPv6 SRH as a socket option when creating TCP connections. M1 can be configured either as a router or with our transparent proxy. When used as a router, we create static routes and use the standard Linux IPv6 forwarding.

A. Maximum throughput

First, we compare the performance of one of our proxies against the performance of a Linux router running on the same platform. In this setup, our client uses `wrk` [26] to simulate 200 web client downloading static web pages of given sizes during 120 seconds. It uses 8 threads with 25 connections per thread. The proxy was configured with a virtual function that directly copies that bytestreams without any processing.

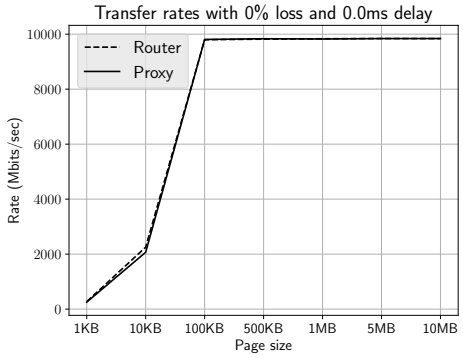


Fig. 6: Raw throughput.

Figure 6 shows the total transfer rate when the client is downloading web pages. This figure shows that there is no significant difference in transfer rates between our proxy and the router. With 10MB files, our proxy reaches a throughput of 9841 Mb/s where the router achieves 9838 Mb/s. A closer look at the small page sizes in figure 6, shows that our proxy slightly underperforms the router. With 1KB files, our proxy achieves a rate of 253 Mb/s, while the router achieves a rate of 272 Mb/s.

In term of requests per second, for 1KB files, our proxy completes 26634 requests per second, while the router completes 28613 requests per second. This difference in performance between large and small files can be explained by the fact that when our proxy receives a new connection from the client, it needs to establish a new connection to the server before starting to forward packets. With smaller files, there are significantly more three-way handshakes to perform, making this overhead more important while this cost is amortized for larger files. With 100KB files, the number of requests per seconds is already on par at ≈11945 requests per second for both the proxy and the router.

B. Impact of packet losses and latency on the proxies

The previous section explored the maximum rate that our proxies can sustain. In those measurements, the TCP stack running on M1 did not have to buffer packets or handle retransmissions. As those operations can affect its performance, we added `netem` to simulate different delays and different packet loss ratios.

We start by adding a 1% loss and a 25ms delay on the four links of figure 5. This corresponds to an end-to-end loss of ≈4%, and an end-to-end latency of 100ms. The results of this measurement are shown on figure 7. Under such circumstances, our proxy outperforms the router. This is not surprising since in this setup, our proxy acts as a Performance Enhancing Proxy (PEP). While figure 7 clearly shows a large improvement for large file sizes, our measurements indicated that this is also true for small file sizes. This can be explained by the fact that when M1 is configured as a router all packet losses need to be recovered end-to-end. When a packet is lost on the same link with our proxy, the retransmission is done by the proxy.

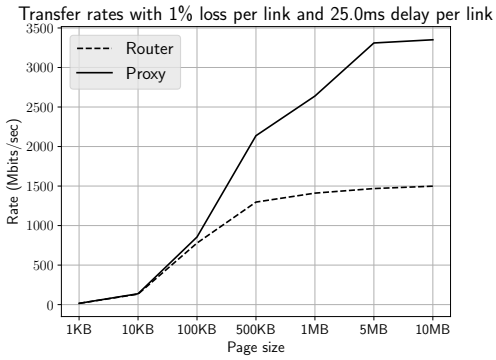


Fig. 7: Transfer rate with 1% of loss and 25ms of latency per link.

To confirm our findings, we run the same measurement, but adding latency and loss only on the link between the server and the proxy, the objective being to mimic a network where the loss would happen only on the link between the proxy and the server. To replicate our previous configuration, we add 2% of loss per link, to get an end-to-end loss of ≈4%, and 50ms of latency per link to get an end-to-end latency of 100ms. As shown by figure 8, under such conditions, the proxy and the router are both significantly affected by the performance degradation in the same fashion, confirming our findings.

C. CPU-intensive Virtual Functions

With our architecture, various types of Virtual Functions can be implemented. Some like a PEP simply proxy the connections and do not need to process the payload. Others like DPIs, transparent compression or transparent encryption need to process the payload and thus consume CPU cycles. To

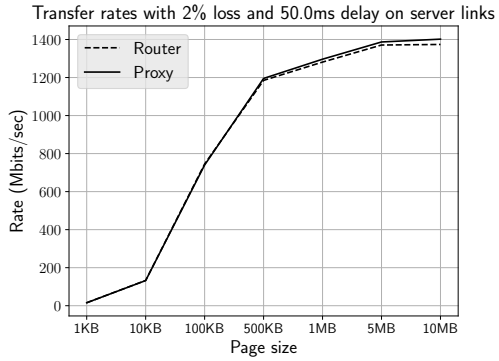


Fig. 8: Transfer rate with 2% of loss per link and 50ms of latency per link between the proxy and the server.

measure the impact of the Virtual Function on the performance of our proxy, we developed a simple microbenchmark that performs $2 \times n$ passes over the bytestream and XORs each byte with a key at each pass. This VF leaves the bytestream unmodified, but consumes both CPU and accesses memory.

The results with this microbenchmark are shown in figure 9. When our VF performs two passes on the bytestream, the maximum throughput is similar to the one we obtained without bytestream modification in figure 6. When the VF performs four passes on the bytestream, the maximum throughput with pages larger than 100KB is divided by 2. This throughput continues to drop with the CPU load on the VF. To confirm that the reduction in throughput was due to the CPU intensive computations, we ran `perf` [27] that yielded 96% of cycles spent in the XOR function.

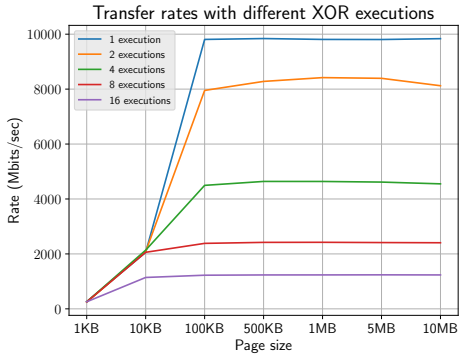


Fig. 9: Maximum throughput with Virtual Functions performing n passes over the bytestream.

D. Chaining middleboxes

With our architecture, middleboxes can be used in chains where one middlebox performs the opposite function of the previous one. Typical examples include transparent compression/decompression or transparent encryption. To demonstrate this use case, we implemented a VF that simply XORs each byte of the bytestream with a constant. When two such middleboxes are used in sequence, the bytestream output of the downstream one is the same of the input of the upstream one. This is illustrated in figure 10.

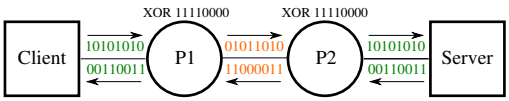


Fig. 10: Demonstration of middlebox chaining with simple XOR transformations.

Due to limitations of our lab, we could only perform this experiment over 1 Gbps links. Figure 11 shows that with the two chained middleboxes, the maximum throughput was the same as when passing through two routers. This is expected given the results of figure 9 with 10Gbps interfaces.

Fig. 11: Transfer rate of wrk with 2 proxies applying a XOR.

VI. Related Work

AbdelSalam et al. propose in [28] to use IPv6 Segment Routing to support Virtual Network Function Chaining and implement a prototype as a Linux kernel module. They leverage namespaces to support virtual network functions but only support packet-based functions while our solution leverages the Linux TCP stack to provide a bytestream abstraction to the network functions. In FlowOS, Bezahaf et al. [21] proposed a Linux kernel module that exposes a bytestream abstraction to network functions but they do not describe how flows are routed through the network functions. NetVM [29] leverages virtualization techniques and a user-space packet processing platform to provide fast, chainable network functions in virtual machines. Their work focuses on packet processing and does not consider bytestream functions. Other solutions such as XOMB [30] focus on the system aspects of implementing virtual functions to support middleboxes through a flexible programming model. Our architecture leverages IPv6 Segment Routing to forward the packets to the middleboxes. Another related work is `/dev/stdpkt` proposed by Utsumi et al in [31]. `/dev/stdpkt` uses the Linux Kernel Library to implement virtual functions that can be chained together.

VII. Conclusion

Given its ability to enforce precise network paths for specific flows, IPv6 Segment Routing appears to be an excellent

candidate to support middleboxes in entreprise networks. We leverage this IPv6 extension in our architecture designed for enterprise networks. Its main benefit is that the middleboxes are explicitly exposed. This significantly improves the manageability of the network. Our architecture supports both middleboxes that operate on a per-packet basis (e.g. NAT, stateless firewalls) and those that need to process bytestreams (e.g. DPI, Application Level Gateways, ...). For the latter, we use transparent TCP proxies that process the IPv6 Segment Routing Header. We implement[3] this architecture in the Linux kernel and evaluate its performance with various benchmarks in our lab. Our measurements indicate that our architecture is well suited to support middleboxes that process bytestreams.

VIII. Future Work

In this paper, we implemented a proof of concept using the regular Linux mechanisms. While kernel bypass techniques such as DPDK or user-space TCP stacks like mTCP allow significant performance boosts, they are often specific to a subset of network hardware. By leveraging the kernel datapath, our solution remains generic and can be deployed on any Linux-supported hardware, ranging from high-end servers to home routers. Should an operator require performance only available through kernel bypass techniques, our high-level network architecture would remain identical and our userspace implementation of the proxy would require minimal changes to plug-in with a DPDK-like library. These modifications can be realized as future work.

IX. Acknowledgements

This work was partially supported by the ARC-SDN project and a Cisco URP grant.

References

[1] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), February 2002.

[2] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[3] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle: a field study of middlebox failures in datacenters. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 9–22. ACM, 2013.

[4] Kaustubh Joshi and Theophilus Benson. Network function virtualization. *IEEE Internet Computing*, 20(6):7–9, 2016.

[5] J. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. RFC 7665 (Informational), October 2015.

[6] P. Quinn, U. Elzur, and C. Pignataro. Network Service Header (NSH). Internet draft, draft-ietf-sfc-nsh-28, November 2017.

[7] Mehdi Nikkhah and Roch Guérin. Migrating the internet to ipv6: an exploration of the when and why. *IEEE/ACM Transactions on Networking*, 24(4):2291–2304, 2016.

[8] Mat Ford. Landmark ipv6 report published: State of deployment 2017. CircleID, http://www.circleid.com/posts/20170606_landmark_ipv6_report_published_state_of_deployment_2017/, June 2017.

[9] Clarence Filsfils et al. The segment routing architecture. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[3]To ensure the reproducibility of our results, our implementation and the measurement scripts will be released on http://segment-routing.org/index.php/SRv6Pipes at publication time.

[10] Stefano Previdi et al. IPv6 Segment Routing Header (SRH). Internet-Draft draft-ietf-6man-segment-routing-header-07, Internet Engineering Task Force, July 2017. Work in Progress.

[11] Christian Kreibich, Mark Handley, and V Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. USENIX Security Symposium*, volume 2001, 2001.

[12] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[13] Olivier Bonaventure and SungHoon Seo. Multipath TCP deployments. *IETF Journal*, 2016, November 2016.

[14] Olivier Bonaventure, Mohamed Boucadair, Bart Peirens, SungHoon Seo, and Anandatirtha Nandugudi. 0-RTT TCP Converter. Internet-Draft draft-bonaventure-mptcp-converters-02, Internet Engineering Task Force, October 2017. Work in Progress.

[15] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 178–189. ACM, 1998.

[16] Jun Xin, Chia-Wen Lin, and Ming-Ting Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1):84–97, 2005.

[17] Amazon Elastic Transcoder. https://aws.amazon.com/fr/elastictranscoder/. Accessed: 2018-04-05.

[18] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment Routing Architecture. Internet-Draft draft-ietf-spring-segment-routing-14, Internet Engineering Task Force, December 2017. Work in Progress.

[19] David Lebrun. *Reaping the Benefits of IPv6 Segment Routing*. PhD thesis, UCLouvain / ICTEAM / EPL http://hdl.handle.net/2078.1/191759, October 2017.

[20] David Lebrun, Mathieu Jadin, François Clad, Clarence Filsfils, and Olivier Bonaventure. Software resolved networks: Rethinking enterprise networks with ipv6 segment routing. In *SOSR'18: Symposium on SDN Research*, 2018.

[21] Mehdi Bezahaf, Abdul Alim, and Laurent Mathy. Flowos: A flow-based platform for middleboxes. In *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '13, pages 19–24, New York, NY, USA, 2013. ACM.

[22] Bhavish Agarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, and George Varghese. Endre: An end-system redundancy elimination service for enterprises. In *NSDI*, pages 419–432, 2010.

[23] Clarence Filsfils, Francois Clad, Pablo Camarillo, Jose Liste, Prem Jonnalagadda, Milad Sharif, Stefano Salsano, and Ahmed AbdelSalam. Ipv6 segment routing. In *SIGCOMM'17, Industrial demos*, August 2017.

[24] David Lebrun and Olivier Bonaventure. Implementing IPv6 Segment Routing in the Linux Kernel. In *Proceedings of the 2017 Applied Networking Research Workshop*. ACM, July 2017.

[25] P. Camarillo et al. Srv6 network programming. Internet draft, draft-filsfils-spring-srv6-network-programming-02, work in progress, October 2017.

[26] wrk - a HTTP benchmarking tool. https://github.com/wg/wrk. Accessed: 2017-12-31.

[27] perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/. Accessed: 2018-03-29.

[28] Ahmed AbdelSalam, Francois Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, and Luca Veltri. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In *IEEE Conference on Network Softwarization (NetSoft)*, Bologna, Italy, July 2017.

[29] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 445–458, Berkeley, CA, USA, 2014. USENIX Association.

[30] James W. Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. xomb: Extensible open middleboxes with commodity servers. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '12, pages 49–60, New York, NY, USA, 2012. ACM.

[31] Motomu Utsumi, Hajime Tazaki, , and Hiroshi Esaki. `/dev/stdpkt`: A service chaining architecture with pipelined operating system instances in a unix shell. In *AINTEC '17: Asian Internet Engineering Conference*, Bangkok, Thailand, November 20–22 2017.

SERA: SEgment Routing Aware Firewall for Service Function Chaining scenarios

Ahmed Abdelsalam*, Stefano Salsano†, Francois Clad‡, Pablo Camarillo‡, Clarence Filsfils‡,
*Gran Sasso Science Institute, †University of Rome Tor Vergata, ‡Cisco Systems

***Abstract*—In this paper we consider the use of IPv6 Segment Routing (SRv6) for *Service Function Chaining* (SFC) in an NFV infrastructure. We first analyze the issues of deploying *Virtual Network Functions* (VNFs) based on SR-unaware applications, which require the introduction of SR proxies in the NFV infrastructure, leading to high complexity in the configuration and in the packet processing. Then we consider the advantages of SR-aware applications, focusing on a firewall application. We present the design and implementation of the SERA (SEgment Routing Aware) firewall, which extends the Linux iptables firewall. In its basic mode the SERA firewall works like the legacy iptables firewall (it can reuse an identical set of rules), but with the great advantage that it can operate on the SR encapsulated packets with no need of an SR proxy. Moreover we define an advanced mode, in which the SERA firewall can inspect all the fields of an SR encapsulated packet and can perform SR-specific actions. In the advanced mode the SERA firewall can fully exploit the features of the IPv6 Segment Routing network programming model. A performance evaluation of the SERA firewall is discussed, based on its result a further optimized prototype has been implemented and evaluated.**

***Index Terms*—Service Function Chaining (SFC), NFV, Segment Routing, Linux networking, Firewall, Iptables**

I. Introduction

The advent of Network Function Virtualization (NFV) [1] is dramatically changing the way in which telecommunication networks are designed and operated. Traditional specialized physical appliances are replaced with software modules running on a virtualization infrastructure made up of general purpose servers. Such virtualization infrastructure can even be composed of a set of geographically distributed data centers. In traditional "pre-NFV" networking, the physical appliances were placed *en-route*, i.e. along the path of the flows. In NFV scenarios, the Virtual Network Functions (VNFs) that replace the physical appliances can be arbitrarily located in the distributed virtualization infrastructure, hence the need of steering the traffic flows through the sequence of VNFs to be accessed. The VNFs can also be denoted as Service Functions (SF) and the *Service Function Chaining* (SFC) [2] denotes the process of forwarding packets through the sequence of VNFs. Examples of VNFs categories are NATs (Network Address Translation), firewalls, DPIs (Deep Packet Inspection), IDSs (Intrusion Detection System), load balancers, HTTP proxies, CDN nodes.

This work has been partially supported by the Cisco University Research Program Fund

ISBN 978-3-903176-08-9

The IETF SFC Working Group (WG) has investigated the SFC scenarios and issues [3] and proposed a reference architecture [4]. A specific mechanism, called Network Service Header (NSH) [5] has been proposed by the SFC WG to support the encapsulation of packets with a header that specifies the sequence of services (VNFs) to be crossed.

In this paper, we consider the use of the IPv6 Segment Routing (SRv6) architecture to support Service Function Chaining, as already discussed in [6]. In the SRv6 architecture an IPv6 extension header (the Segment Routing Header - SRH) allows including a list of *segments* in the IPv6 packet header [7]. This segment list can be used to steer the packet though a set of intermediate steps in the path from the source to the destination, following a (loose) source routing approach. The use of Segment Routing for SFC has been documented in [8]. The typical network scenario is that an edge node classifies the traffic and consequently includes a segment list in the IPv6 packet header. Note that the application of SRv6 is not limited to SFC, there are many other important use cases [9] like for example traffic engineering, fast restoration, support of Content Delivery Networks. The concept of SRv6 has been extended in [10], from the simple steering of packets across nodes to a general *network programming* approach. The idea is to encode *instructions* and not only *locations* in a segment list. This is feasible, thanks to the huge IPv6 addressing space. Under this network programming model, the edge node can *program* a sequence of nodes to be crossed and the packet processing/forwarding behaviors to be executed by the nodes on the packet.

In section II we discuss how the IPv6 Segment Routing can be used to support SFC, what are the implications of using SR-unaware applications and the potential advantages of having SR-aware applications. As we will extend the open source Linux iptables firewall, section III provides a short introduction to its architecture. In section IV we analyze some design requirements and use case scenarios for the SR-aware applications, focusing on a firewall application. An important contribution here is the inclusion of a scenario in which some instructions to the firewall (e.g. actions to be executed on some class of packets) can be included in the segment list associated to a packet, without the need of reconfiguring the rules in the firewall running in the core of the NFV infrastructure and in line with the SRv6 network programming approach [10]. From the requirements, we design the architecture of the proposed SEgment Routing Aware (SERA) firewall, which extends the iptables firewall. In section V some details of the implemen-

tation are given. To the best of our knowledge, the SERA firewall can be considered the first SRv6-aware application. Section VI provides the description of the testbed and the result of the performance evaluation. Based on these results, we have identified some shortcomings of the iptables design for our use cases. We implemented and evaluated a proof-of-concept that shows a significant performance improvement.

II. SFC BASED ON IPV6 SEGMENT ROUTING

Following the terminology defined in [4], the *SFC encapsulation* carries the information to identify the sequence of Service Functions (VNFs) that are required for processing a given packet. In the SRv6 approach considered here, the IPv6 Segment Routing Header (SRH) [7] contains such information. The SRH contains a *segment list*, a segment in this list identifies a VNF. Moreover, additional information related to the VNF chain can be carried in the optional Tag-Length-Value (TLV) section at the end of the SRH.

When a VNF that processes the packets is a legacy VNF, which is not aware of the Segment Routing based SFC encapsulation, we refer to it as an *SR-unaware* application. In this case an *SFC proxy* is needed, to remove the SFC encapsulation and deliver a clean IP packet to the SR-unaware application. Considering our focus on the Segment Routing based solution, we refer to the SFC proxy as *SR-proxy*. For the packets that are sent by the SR-unaware application, the SR proxy needs to (re)apply the SFC encapsulation after proper classification of the received packets. The operations of the SR proxies tend to be complex and in general they are not efficient. The main issue is that the information contained in the SFC encapsulation is removed from the packet when the packet is delivered to the SR-unaware application and may need to be re-added to the packet. This process typically requires a lot of state information to be configured in the classifier components of all nodes of a VNF chain and can consume a considerable amount of packet processing resources in the nodes.

Different types of VNFs can process the IP packets in a VNF chain. Some VNFs only need to inspect IP packets (e.g. DPI or network monitoring applications), other can drop or admit packets (e.g. firewall), other can modify IP and transport layer headers (e.g. NATs), other may need to terminate transport layer connections and reopen new ones (e.g. HTTP proxies, TCP optimizers). In general, the operations of an SR proxy depend on the type of VNF. If the VNF is not operating on the connections at transport layer (i.e. it is not modifying the 5-tuple of IP and transport layer source and destination addresses) it is possible in principle to re-classify the packets, at the price of repeating the flow-level classification in all nodes of the VNF chain. If the VNF is terminating / opening new transport level connections, it is not always possible to re-classify the packets and associate them to a specific chain.

As described in the SRv6 network programming document [10], the SR information can be added to a packet in two different modes, insert or encap. Figure 1 shows the original IPv6 packet and how it is carried in the two different encapsulation modes. In the insert mode the SRH header is inserted in the original IPv6 packet, immediately after the IPv6 header and before the transport level header. Note that the original IPv6 header is modified, in particular the IPv6 destination address is replaced with the IPv6 address of the first segment in the segment list, while the original IPv6 destination address is carried in the SRH header as the last segment of the segment list. In the encap mode the original IPv6 packet is carried as the inner packet of an IPv6-in-IPv6 encapsulated packet. The outer IPv6 packet carries the SRH header with the segment list.

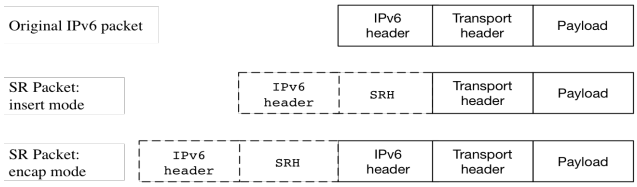


Fig. 1: SRv6 encapsulated packets

An SR-unaware application is not able to process the SRH information in the traffic it receives. An SR proxy is used to process the SRH information on behalf of the SR-unaware application. As discussed above, the behavior and the applicability of an SR proxy depend on the type of processing of the application. In [8], a set of behaviors of the SR-proxy have been defined, among them we mention:

- Static proxy
- Dynamic proxy
- Masquerading proxy

Both the Static and the Dynamic proxies support IPv6 SR packets encapsulated only in encap mode. They remove the SR information from packets before sending them to an SR-unaware application. These proxies receive back the packets from the SR-unaware application and reapply the SR encapsulation which includes the information on the VNF chain. They work under the assumption that the specific SR-unaware application running in the VNF is inserted in only one VNF chain, because all packets going out from the VNF are re-associated to the same chain. If multiple VNF chains needs to be supported, a different instance of the VNF is needed for each chain. The difference between the Static and the Dynamic proxies is that the SR information is statically configured in the Static case and is read from the incoming packets in the Dynamic case.

The Masquerading proxy supports SR packets encapsulated in insert mode. It masquerades SR packets before they are sent to an SR-unaware application, replacing the IPv6 destination address (which correspond to the current segment of the segment list) with the original IPv6 destination (i.e. the last segment in the segment list). When the packets are received back from the SR-unaware application, the Masquerading proxy retrieves the VNF chain information from the SRH header and changes the IPv6 destination address so that it reflects the current segment of the segment list. This process

is refered to as *de-masquerading*. The assumption is that the SR-unaware application simply ignores the SRH header and that the SRH header is preserved in the processing. Moreover, this type of proxy can be only used with applications that do not change the packet headers and just inspect them.

Following the above discussion on the SR-unaware applications, we can state that their use in combination with SR proxies is conditioned by some constraints and characterized by high configuration complexity. It can also be affected by performance issues. Of course these problems will be faced and solved in practical use cases, considering the importance to support legacy SR-unaware applications in NFV deployments. On the other hand, in this work we take a more forward-looking approach and consider the design and development of SR-aware applications. Such applications are able to process the SFC encapsulation included in the IP packets, that is in our scenario the IPv6 SRH header that contains the segment list. The greatest benefit of using SR-aware applications is that the SR proxy is not needed and the SFC information carried in the SRH header is preserved when the packet is processed by the application. This approach avoids the need to maintain state information in the internal nodes. The configuration and management of the NFV infrastructure is simplified and the performance of the NFV enabled nodes is not affected by complex classification procedures. Moreover advanced features are possible by letting the applications interact with the SFC functionality offered by the network.

In this paper, we focus on the design and implementation of an SR-aware firewall application, but most of the design considerations have a more general applicability to other types of applications that can be deployed in SR based Service Function Chaining scenarios.

III. Linux iptables firewall

A firewall [11] essentially works according to a set of rules to accept or drop received packets. Each rule is composed of a condition and an action. The condition is based on set of attributes of received packets.

Once a packet satisfies the condition expressed by a rule condition, the associated action is performed on that packet. Iptables is a flexible and modular firewall and it is a standard component of most Linux distributions. It is built on top on the netfilter framework. In this section we provide a short tutorial on Iptables and netfilter architecture and implementation, which will be the base for the design of our solution.

A. Netfilter Framework

The netfilter framework [12] is a set of hooks in the packet traversal through the Linux protocol stack, which allows access to packets at different points. The current netfilter implementation provides five different hooks (*PREROUTING*, *INPUT*, *FORWARD*, *OUTPUT*, *POSTROUTING*) distributed along the receive and transmit path of packets as shown in Fig. 2. Kernel modules can register callback functions at any of these hooks. A callback function, after processing a packet, returns to the netfilter hook the action to be taken on the packet, such as DROP, ACCEPT, QUEUE (queue for user space processing).

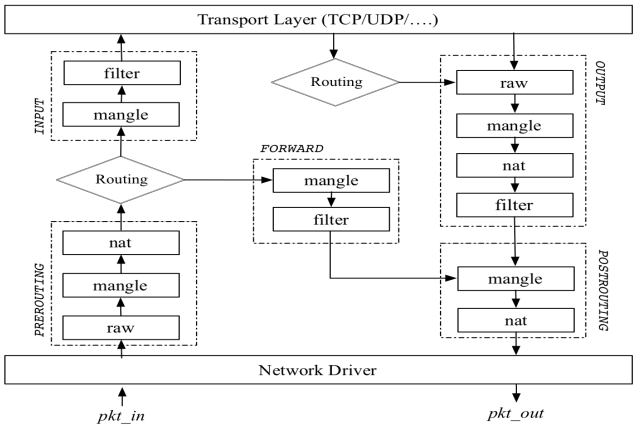


Fig. 2: Netfilter hooks and their associated tables

B. Iptables

Iptables represents the userspace implementation which allows access to the kernel-level netfilter framework hooks. It defines a set of rules that instruct the kernel what to do with packet coming to or traversing the protocol stack. The implementation of netfilter includes some pre-defined tables, as shown in Figure 2. Each table has a set of chains where iptables rules can be inserted. The supported tables are: filter (the default table, it contains rules that are used to filter IP packets); nat (mainly used to re-write the source and/or destination addresses of IP packets); mangle (a specialized table for mangling packet as they go through the kernel); raw (mainly used for connection tracking). Each iptables rule defines a set of matching criteria based on information from different layers of the protocol stack. Once the packet matches a rule, iptables takes an action on this packet. The standard actions are: ACCEPT, DROP, or QUEUE. Those correspond to the callback functions return values. Listing 1 shows examples of iptables rules.

C. Iptables extensions

The iptables framework is modular and extendible. New match extensions and target extensions can be developed separately and added to the iptables as new modules.

Listing 1: Examples of iptables rules

```
# Standard iptables rule
# Matches destination address of a packet
ip6tables -I INPUT -d fc00:d1::/64 -j DROP

# Extended iptables rule
# Matches destination address and hop-by-hop header
ip6tables -I INPUT -d fc00:d1::/64 \
-m hbh --hbh-len 40 -j DROP
```

Match extensions are used to add more matching options to iptables. They can be used alone or in combination with the default match options. They provide the ability to have sophisticated iptables rules in order to look deeper into IP packets. e.g., hbh, which matches the parameters in IPv6 Hop-by-Hop extensions header. An example of extended iptables rule is shown in Listing 1.

Target extensions are new actions added to the default ones of iptables. A new iptables target usually performs an action different from the default ones (ACCEPT, DROP, etc.,). It can be used for logging/profiling or it can modify the packet before returning it back to the netfilter framework. Destination NAT (DNAT) is an example of iptables target extension, which is used to modify the destination address of a packet.

IV. SEgment Routing Aware (SERA) firewall

In an SRv6 SFC scenario, the VNFs are deployed over the servers of the NFV infrastructure. The Segment Routing Header (SRH) is added to packets to enforce a VNF chain, i.e. the sequence of VNFs to be crossed by the packets. The SR-unaware applications rely on the SR-proxy that removes the SRH from the packet. On the other hand, the SR-aware applications are capable of processing the SR information in the packets. We focus on a specific type of SR-aware applications, namely a firewall application. In this section, we start by analyzing some design requirements and use case scenarios for the SR-aware applications. The following considerations are focused on a firewall application, but they have a more general value as they can be applied to similar applications that needs to be deployed on an SR based SFC environment (e.g. DPI, IDSs). We assume that an SR-aware firewall should support two working modes: *basic* mode and *advanced* mode.

In the basic mode the SR-aware firewall must be able to work as a legacy firewall, but with no need of the SR-proxy. In particular, the SR-aware firewall should be able to use the same set of rules defined for the legacy firewall and apply them directly to the SFC encapsulated packets that carry the SRH information. It must be able to handle SR packets encapsulated in encap as well as insert modes and logically apply the rules to the original packets rather than to the SFC encapsulated packets. To make a concrete example, if an existing rule includes a condition on the source IPv6 address and the original IPv6 packet has been encapsulated in (IPv6-in-IPv6) it makes no sense to consider the IPv6 source address of the received packet as the condition should be checked on the source address of the packet. The use case scenario is to virtualize the legacy firewalls, executing them in servers on the NFV infrastructure, without changing the legacy rules and with no need of SR-proxy functionality.

In the advanced mode the SR-aware firewall should support rules with extended conditions that can explicitly include attributes not only from the original packet but also from the SRH and the outer packet. In particular, the SR-aware firewall could leverage SRv6 SID arguments, TLVs, or TAG. It could also apply differentiated processing based on the active SRv6 SID (i.e., apply different rule sets for different SIDs). As for the actions, in the advanced mode the SR-aware firewall should be able to support some SR-specific actions. For example, an SR-specific action could be to skip the next SID in the segment list, so that it is possible to operate a "branching" instead of the usual linear exploration of the VNF chain, when some conditions on the packet are met. A use case scenario for this feature is to consider a service chain which includes a firewall followed by an Intrusion Detection System and allow skipping the IDS for a subset of traffic that matches some conditions. A further requirement is that the SR-aware firewall application should be able to select the actions to be performed based on information contained in the SID. This is aligned with the SRv6 network programming approach of minimizing the state information maintained in the nodes and storing explicit state information in the packets. The use case scenario in this case is that instead of re-configuring some firewall rules in a specific firewall running in the core of the NFV infrastructure, it is possible to obtain the same result by changing a SID in the SID list that is injected to the packet in the edge node. The big advantage is that the reconfiguration is only needed in the edge node, which in any case has to manage per-flow state to perform the classification operations.

In the following subsections we propose the architecture of the SERA firewall (for basic and advanced modes) that meets the above requirements, extending the Linux iptables.

A. SERA basic mode

In the basic mode, SERA is an SR-aware firewall that can apply the normal firewall processing to the original packets even if they have an SR based SFC encapsulation. The proposed packet processing architecture is shown in Figure 3. Each received packet goes through an *SR pre-processor* that splits traffic into SR and non-SR traffic. Non-SR traffic is processed as in an SR-unaware firewall, as represented with the solid-line path in Figure 3. SR traffic follows a different path through the firewall, represented with double-line path in Figure 3. In this path, the firewall evaluates the defined rules on the original packet, properly taking into account the impact of the SR encapsulation. It supports both encap and insert mode, which implies that the original IPv6 source and destination information of received packets may be encoded differently as follows:

- Encap mode: original source and destination are the ones of the packet.
- Insert mode: packets have only one IPv6 header. The original source information is in the source address of the IPv6 header, while the original destination is encoded as the last SID in the SRH.

The *Inner match* functional block is responsible for getting the original source and destination information from SR packets and compare them to the defined rules. Once a packet hits a condition of a rule, the associated standard action (ACCEPT, DROP, etc.) is triggered on that packet.

B. SERA advanced mode

In the advanced mode, SERA extends the iptables capabilities by offering new matching capabilities and new SR-specific actions. It introduces new iptables rules (SERA rules) that have extended conditions involving attributes from outer packet, inner packet, and the SRH header. The architecture of advanced mode (Figure 4) is defined incrementally with respect to the basic mode (Figure 3), by adding the *SRH match* functional block and replacing the *Action* block with the *Extended Action* block. Since the matching could be performed on both the original and the outer packet headers, the SR traffic follows a more complex path, as shown in Figure 4. Unlike in the basic mode SERA, all received packet are first processed by the *Outer match* block, which applies parts of the extended rules on the outer packet. The *SR pre-processor* does the same job as in the basic mode SERA by splitting traffic into non-SR and SR traffic. Non-SR traffic goes directly to the Action functional block while SR traffic is directed to the the *Inner match* block. The Inner match block works as in the basic mode, but the rules that drive its behavior are written in a different way. For example, with an extended rule it is possible to match on the outer source and destination IPv6 addresses (denoted as `src`, `dst`) and on the original ones (denoted as `inner-src`, `inner-dst`). The Inner match block takes care of the matching of the inner source and destination (the ones of the original packet). The *SRH match* block is concerned with the matching between SRH extension part of the rules and the SRH of received SR packets. Finally, each packet (SR or non-SR) that satisfies the matching condition of a rule goes to the *Extended Action* block. It extends the Action block present in the architecture of the Basic mode by allowing the introduction of *SR-specific* actions in addition to the standard ones.

An SR-specific action is an advanced action that can be applied to SR-encapsulated packets. It may modify or process SR-encapsulated packets based on SRH information. We list here some examples of SR-specific actions, but the set of these actions can be extended to cover more complex SFC use-cases.

- `seg6-go-next`: the default action of the SEG6 target. It is similar to the Endpoint function from the SRv6 network programming model [10]. It sends packets towards the next SID from SRH. The `seg6-go-next` serves as an ACCEPT action for SRv6 encapsulated packets.
- `seg6-skip-next`: it instructs the SERA firewall to skip the next SID in the SRH.
- `seg6-go-last`: it instructs the SERA firewall to skip the remaining part of the segment list and process the last segment.
- `seg6-eval-args`: the generic action that supports SRH programmed actions.

Following the traditional iptables model, the above defined SR-specific actions are included in *statically* configured rules which are executed in a SERA firewall running as a VNF. Taking into account the concepts of the SRv6 programming model, we have designed a more dynamic approach, which

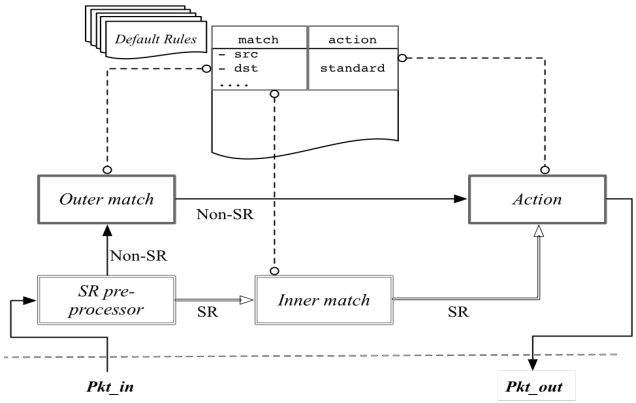


Fig. 3: Architecture of basic mode SERA

allows to define the action to be executed as a result of a match on a packet by packet basis, by putting information in the Segment IDentifier (SID). For this purpose, a special SR-specific action is defined, called `eval-args`. It does not represent a concrete action, but instructs the SERA firewall to look into the current SID to find the action to be executed. As described in [10], an SRv6 local SID is an IPv6 address that can be logically split into three fields: LOC:FUNCT:ARGS. LOC uses the L most significant bits, ARGS the R rightmost-bits and FUNCT the remaining $128 - (L + R)$ bits in the middle. In our case, the LOC part is used as a locator to forward the packets to the NFV node that runs the firewall, and it is advertised by the routing protocols. The FUNCT part identifies a specific VNF on the NFV node (in our case the SERA firewall instance). The ARGS part may contain information required by the VNF and may even change on a per-packet basis. Note that the ARGS part will be ignored in most cases (or omitted setting R=0), whenever there is no need to carry additional information in the SID. To give an example, the LOC field can be 64 bits long and uniquely identify an NFV node. This leaves 128-64=64 bits for the identification of the VNF in the NFV node and for the arguments if needed.

In the advanced mode of SERA it is possible to use the ARGS part of the SID to encode a firewall action to be executed in case of match. This requires that a set of rules with action `eval-args` is configured in the SERA firewall. For all packets that match one of these rules, the action to be executed is contained in the ARGS field of the SID. The advantage of this approach is that it is possible to (re)configure the action to be executed on a given subset of packets by operating at the network edge, with no need to update the configuration of the SERA firewall instance running in the core of the NFV infrastructure.

V. Implementation

We implemented the SERA firewall as an extension of Linux iptables described in section III.

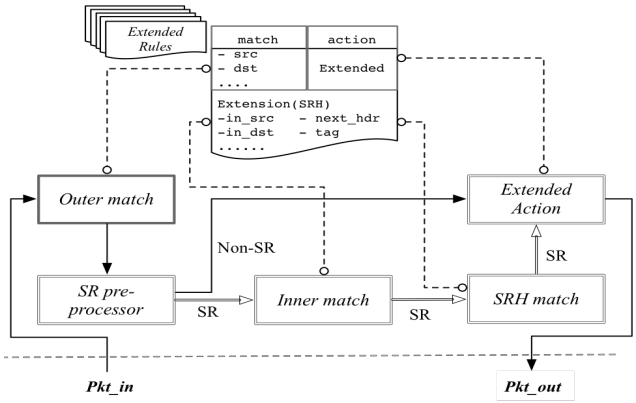


Fig. 4: Architecture of advanced mode SERA

A. Implementation of basic mode SERA

In the Linux kernel the `ip6_tables` module is responsible for checking the iptables rules against the received packets. It implements the `ip6_packet_match()` function that evaluates the defined iptables rules against the outermost IPv6 header of a received packet. In order to implement the basic mode of the SERA firewall, we extended the existing `ip6_tables` module to operate according to the architecture shown in Figure 3. We added the SR pre-processor block. The SR packets are forwarded to the *Inner match* functional block, implemented in the `inner_match()` function, which evaluates iptables rules against the original packet. It supports SR packets encapsulated in both encap and insert mode.

We added a new sysctl parameter (`ip6t_seg6`) to switch between legacy iptables mode and SERA basic mode. The system administrator can enable the SERA basic mode on the fly with the command: `sysctl -w net.ipv6.ip6t_seg6=1`, which activates the SR pre-processor.

We have realized a first version of basic mode SERA that implements only a subset of the normal classification rules, namely those involving the IP src and destination addresses. On this first version we have performed the evaluation that is reported in the paper. Then we have implemented a second version that supports all the classification rules and it is now available at [13].

B. Implementation of advanced mode SERA

We implemented the advanced mode SERA by exploiting the iptables extension features. We added a new match extension as well as a new target extension to the iptables implementation both at kernel and user-space levels. Thanks to these extensions it is possible to match on the SRH fields, this allow to have a full control on where the packets is directed (the next SIDs) and which nodes it has crossed before.

At kernel level, we implemented two additional kernel modules: the `ip6t_srh` as match extension and `ip6t_SEG6` as target extension. The `ip6t_srh` module implements the *SR pre-processor*, the *Inner match*, and the *SRH match* from the advanced SERA architecture. The `ip6t_SEG6` module implements the *Extended Action*. It is a new target (SEG6) for iptables rules that supports a set of SR-specific actions.

To support the advanced mode SERA at user-space level, we extended the iptables user-space utility with two new shared libraries: `libip6t_srh` and `libip6t_SEG6`. They allow the iptables user to define SERA rules. These rules can have attributes from outer packet, inner packet, and SRH. List. 2 shows a list of match options supported by the `libip6t_srh` extension.

The `ibip6t_SEG6` extension supports the new SR (SEG6) target with some SR-specific actions (shown in List. 3). For SRH programmed actions, we introduced a new sysctl variable (`ip6t_seg6_args`) that defines the number of rightmost bits in the active SID to be used as ARGS. The `SEG6` target decodes the ARGS bits to decide which action should be taken on the packet. If the decoded value does not correspond to any of the supported actions, SERA will send back an ICMP Parameter Problem message point to the active SID. Such ICMP message can be used to understand which actions are supported by the firewall.

Listing 2: Options of srh match extension

```
#ip6tables -m srh -h
 srh match options:
  [!] --inner-src       addr[/mask]  Inner packet src
  [!] --inner-dst       addr[/mask]  Inner packet dst
  [!] --srh-next-hdr    next-hdr     SRH Next Header
  [!] --srh-len-eq      hdr_len      SRH Hdr Ext Len
  [!] --srh-len-gt      hdr_len      SRH Hdr Ext Len
  [!] --srh-len-lt      hdr_len      SRH Hdr Ext Len
  [!] --srh-segs-eq     segs_left    SRH Segments Left
  [!] --srh-segs-gt     segs_left    SRH Segments Left
  [!] --srh-segs-lt     segs_left    SRH Segments Left
  [!] --srh-last-eq     last_entry   SRH Last Entry
  [!] --srh-last-gt     last_entry   SRH Last Entry
  [!] --srh-last-lt     last_entry   SRH Last Entry
  [!] --srh-tag         tag          SRH Tag
  [!] --srh-psid        addr[/mask]  SRH previous SID
  [!] --srh-nsid        addr[/mask]  SRH next SID
```

Listing 3: Options of SEG6 target extension

```
#ip6tables -j SEG6 -h
 SEG6 target options:
 [--seg6-action  action]
 Valid SEG6 actions:
 seg6-go-next        SEG6 go next
 seg6-skip-next      SEG6 skip next
 seg6-go-last        SEG6 go last
 seg6-eval-args      SEG6 eval args
```

VI. Performance Evaluation

A. Testbed description

In order to verify the correctness of SERA implementation and to evaluate the performance aspects, we designed a testbed environment that can be easily replicated, shown in

Fig. 5. For the experiments described in this section, we have deployed the testbed on CloudLab [14]. Cloudlab is a flexible infrastructure dedicated to scientific research on the future of cloud computing. Our testbed is composed of three identical nodes. Each node is a bare metal server with Intel Xeon E5-2630 v3 processor with 16 cores (hyper-threaded) clocked at 2.40GHz, 128 GB of RAM and two Intel 82599ES 10-Gigabit network interface cards. The three nodes are Linux servers and respectively represent an ingress node, NFV node, and egress node of an SRv6 based SFC scenario. The links between any two nodes X and Y are assigned IPv6 addresses in the form `fc00:xy::x/64` and `fc00:xy::y/64`. For example, the two interfaces of the link between the ingress node (node 1) and the NFV node (node 2) are assigned the addresses `fc00:12::1/64` and `fc00:12::2/64`. Each node owns an IPv6 prefix to be used for SRv6 local SID allocation. The prefix is in the form `fc00:n::/64`, where `n` represents the node number. For example, the NFV node (node 2) owns the IPv6 prefix `fc00:2::/64`. SRv6 local SIDs are in form LOC:FUNCT:ARGS, where LOC is the most significant 64-bits, ARGS is rightmost 16-bits and FUNCT is the 48-bits in between LOC and ARGS. The ingress node is used as a source for SR encapsulated traffic. The NFV node runs the SERA firewall inside a network namespace. The SERA firewall is instantiated on the SRv6 local SID `fc00:2::f1:0/112`. We have two destination servers $d1$ and $d2$ that are used as traffic sinks. Each destination server is assigned a prefix in the form `fc00:dn::/64`, where `n` is the destination server number. We configured the ingress node with two different SR SFC policies as shown in Listing 4. The first SR SFC policy is used to encapsulate traffic destined to `d1` as SR packets in encap mode, while the second one encapsulates traffic destined to `d2` as SR packets in insert mode. The SRv6 SFC policies are used to steer traffic through the SERA firewall, then to the egress node which removes SR encapsulation from packets as they leave the SR domain towards destinations (`d1` and `d2`). The ingress and egress nodes are running Linux kernel 4.14 [15] and have the 4.14 release of *iproute2* [16] installed. The NFV node runs a compiled Linux kernel 4.15-rc2 with SRv6 enabled and SERA firewall included [17]. In order to saturate

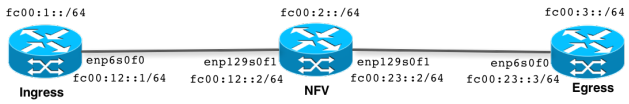


Fig. 5: Performance evaluation testbed.

the CPU of the NFV node, we used only one processor core for processing all the received packets by disabling the *irqbalance* service and assigning the *IRQ* for all interfaces to be served by the same CPU core. We used *iperf* [18] to generate traffic on the ingress node. All traffic generated by *iperf* goes through the SRv6 SFC policies configured on the ingress node.

Listing 4: SR SFC policy

```
# SR SFC policy - encap mode
ip -6 route add fc00:d1::/64 encap seg6 mode \
encap segs fc00:2::f1:0,fc00:3::d6 dev enp6s0f0

# SR SFC policy - insert mode
ip -6 route add fc00:d2::/64 encap seg6 mode \
inline segs fc00:2::f1:0,fc00:3::d6  dev enp6s0f0
```

Listing 5: SR pre-processor implementation

```
static inline bool
sr6_pre_processor(const struct sk_buff *skb,
                  int *innoff,int *srhoff,int *encap)
{
    /* SRv6 traffic (encap mode) detector
    if (ipv6_find_hdr(skb, innoff, IPPROTO_IPV6,
                      NULL, NULL) > 0)){
        *encap=1;
        return true;
    }
    /* SRv6 traffic (insert mode) detector */
    if (ipv6_find_hdr(skb, srhoff,IPPROTO_ROUTING,
                      NULL, NULL) > 0)
        return true;
    return false;
}
```

B. Measurements

In order to evaluate the performance of our implementation, we generated SR traffic with a rate of 1 Mpps (10^6 packets per second). Each packet has a payload size of 1 KB. We wanted to measure the processing capacity (or throughput) of the firewall in processed packets per second (pps). We configured iptables with a rule that drops all traffic going from ingress node towards the destinations. Therefore, the counter of this rule represents the number of SR packets that the firewall has been able to process. In order to evaluate the performance for different numbers of rules, we add a sequence of N-1 non-matching rules before the matching rule. In particular, we repeated each experiment for ten different number of rules N from 1 to 512. Each value plotted in Figures 6-9 represents the average of 30 runs, each run with duration of 60 seconds. The confidence intervals are so close to the average that we have not plotted them.

We conducted five experiments as follows:

- Exp. 1: default iptables on plain IP packets.
- Exp. 2: basic mode SERA with SR encap mode.
- Exp. 3: basic mode SERA with SR insert mode.
- Exp. 4: advanced mode SERA with SR encap mode.
- Exp. 5: advanced mode SERA with SR insert mode.

In experiment 1 (default iptables), we used a rule that matches the IPv6 source and destination address of the received packets. The non-matching rules have the same structure, but different source and destination addresses. With only one rule configured (N=1), the throughput is 911 Kpps. As expected, the achieved throughput decreases with the number

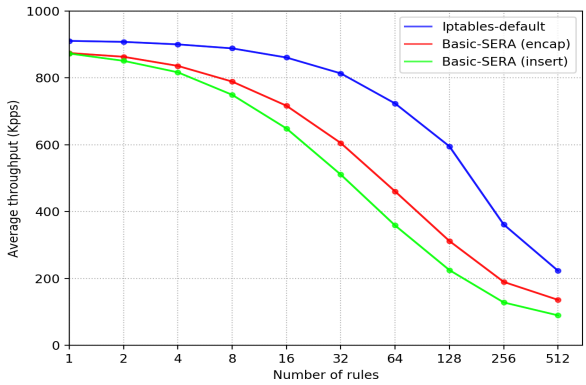


Fig. 6: Basic SERA vs. default iptables

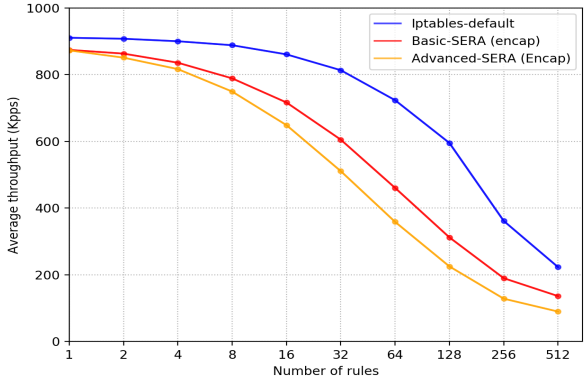


Fig. 7: Basic SERA vs. advanced SERA (encap mode)

of rules, as shown in Figure 6. This is due to the operations that are executed for each rule. In particular, the function `ip6_packet_match()` is called for each rule.

In experiments 2 and 3, we evaluate the throughput of basic mode SERA with the same rules as the ones in the experiment 1 (matching the source and destination address). In these experiments, we are considering SR encapsulated packets and we set the `ip6t_seg6` sysctl to apply the rule to the original packets. When there is only the matching rule ($N = 1$) the throughput is 875 Kpps in encap mode and 873 Kpps in insert mode (Figure 6). For larger N, the degradation of the performance is more evident. The performance reduction of basic SERA with respect to iptables default is due to the *SR pre-processor* functional block, whose implementation is reported in Listing 5. This block has the task to look for the inner IPv6 header in the packet or for the SRH header in case of insert mode (we have re-used the `ipv6_find_hdr` function used by iptables). These operations are computationally expensive and are the reason for the reduction of the throughput visible in Figure 6. According to the design philosophy of iptables, the *SR pre-processor* is executed once for each rule, because each rule operates in a stateless way and no state related to the packet is saved. From a performance point of view, this is clearly not efficient. Therefore, in order to improve the throughput result shown in Figure 6 we are considering alternate design choices which can achieve higher performance when a large number of rules may need to be applied to the packets. The insert mode has lower throughput than the encap mode due to our implementation of the *SR pre-processor* block, which detects SR packets in encap mode before those in insert mode (Listing 5). We decided to add the encap mode detection before the insert mode since it works also for *IPv6-in-IPv6* tunnels.

In experiments 4 and 5, we evaluated the throughput of advanced mode SERA. We considered an extended rule that matches source and destination address from both inner and outer packet. The results are similar to the basic mode SERA, the throughput is 857 Kpps in encap mode and 849 Kpps in insert mode when one rule is configured ($N = 1$) and the performance degradation with respect to the default iptables is higher when the number of rules N increases (the Figure is not reported for space reasons). In Figure 7, we compare the throughput of basic and advanced mode SERA, considering the SR packets in encap mode. Both in the basic and in the advanced mode the *SR pre-processor* is executed once for each rule, the advanced mode SERA achieves a lower throughput because it has to perform two match operations (Inner and Outer) rather than a single one.

We wanted to verify that the throughput reduction when several rules per packet are executed was not caused by problems in our implementation. Hence, we conducted a new experiment using an already existing iptables extension, the Routing Header extension (implemented in `ip6t_rt` kernel module). This extension is able to match the common fields of the IPv6 Routing Header, including the Routing Type field (but is not able to parse the content of the SRH header, for which we have developed the proposed extension). We run the test for the different numbers of rules N as in the previous experiments. For matching we used an extended rule that drops packets with Routing Type 4, i.e. the SR packets with the SRH header. As shown in Figure 8 the obtained throughput perfectly matches our SERA implementation, confirming that the poor performance is inherently related to the iptables design.

Finally, we tackled the issue of performance degradation and we were able to design and implement a solution focusing on one specific scenario, the basic mode SERA operating on SR packets encapsulated in encap mode. In this scenario, a set of existing rules needs to be applied to the original packets that are encapsulated with IPv6-in-IPv6. As shown in Figure 6, there is a performance penalty which becomes significant when the number of rules is large. We revised the design of our iptables extension so that we can execute the SR pre-processor once for each packet instead of re-executing it for every rule. The idea is to modify the pointers that point to the memory area in which the headers of the packet is stored once before executing all the rules and then to properly keep into account these modifications in the processing of the results of the matching. The throughput measurements of the revised design are shown in Figure 9. Only in case of a single

rule, the throughput is slightly reduced do to the operations that are performed once for the packet. When the number of rules increases, there is no throughput degradation as for the basic mode SERA, and the performance approaches the one of the default iptables operating on plain (not encapsulated) IPv6 packets.

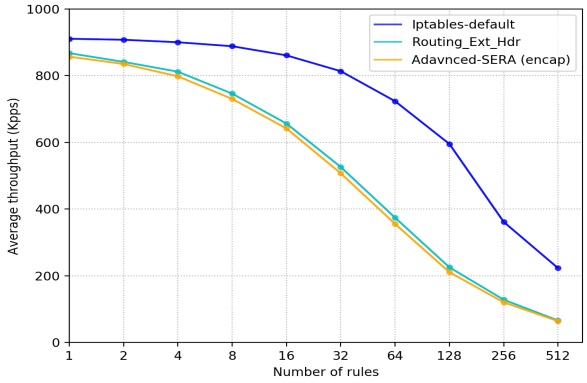


Fig. 8: Existing RH iptables extension vs. advanced SERA

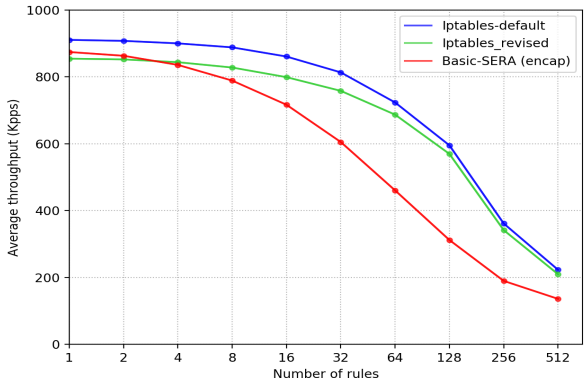


Fig. 9: Revised iptables design vs. Basic SERA

VII. Conclusions

In this work we have shown that it is possible to modify an existing application (the Linux iptables firewall) and make it Segment Routing aware. Thanks to this awareness, it is possible to setup chains of VNFs in a simple and efficient way, with no need of SR proxy. In the basic mode, the proposed SERA firewall solution avoids the need of (re)classification of packets in the intermediate NFV nodes that host the SR-aware firewall. In the advanced mode, new firewall actions can operate on the SR segment list, allowing to make branches in the VNF chain. We have also described how it is possible for the edge node to put instructions in the SR segment list, which can dynamically change the firewall actions that will be executed. This can be done by the edge node even on a packet-by-packet basis. In this way the firewall VNF could become stateless so that it can be scaled, replicated, moved arbitrarily in the NFV infrastructure.

From the performance analysis of the SERA implementation we have highlighted a throughput degradation when the number of rules to be checked for each packet increases. This is due to the iptables design that operates in a stateless way and repeats all operations per each rule. We have implemented a proof-of-concept that overcomes this issue in a specific scenario, showing the performance gain that can be obtained.

We provided an open source implementation for SERA [17]. We submitted our implementation to the the Linux kernel and a part of it has been merged into version 4.16 [13]. We also contributed to the *netfilter.org* project to extend the iptables user-space utility to support the new match options and SR-specific actions (part of our work is in release 1.6.2 of iptables [19]).

References

[1] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[2] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[3] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," Internet Requests for Comments, RFC Editor, RFC 7498, April 2015.

[4] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[5] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," Internet-Draft, November 2017. [Online]. Available: http://tools.ietf.org/html/draft-ietf-sfc-nsh

[6] A. AbdelSalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri, "Implementation of Virtual Network Function Chaining through Segment Routing in a Linux-based NFV Infrastructure," in *3rd IEEE Conference on Network Softwarization (NetSoft 2017), Bologna, Italy*, July 2017.

[7] S. Previdi (ed.) et al., "IPv6 Segment Routing Header (SRH)," Internet-Draft, September 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-02

[8] F. Clad *et al.*, "Segment Routing for Service Chaining," Internet-Draft, October 2017. [Online]. Available: https://tools.ietf.org/html/draft-clad-spring-segment-routing-service-chaining-00

[9] J. Brzozowski et al., "IPv6 SPRING Use Cases," Internet-Draft, December 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-spring-ipv6-use-cases

[10] C. Fisfils *et al.*, "SRv6 Network Programming," Internet-Draft, March 2017. [Online]. Available: https://tools.ietf.org/html/draft-filsfils-spring-srv6-network-programming-04

[11] J. R. Vacca and S. Ellis, *Firewalls: Jump start for Network and Systems Administrators*. Elsevier, 2005.

[12] R. Russell and H. Welte, "Linux netfilter Hacking Howto." [Online]. Available: http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html

[13] "Linux community. linux 4.16 changelog," Web site. [Online]. Available: https://kernelnewbies.org/Linux_4.16

[14] "CloudLab home page," Web site. [Online]. Available: https://www.cloudlab.us/

[15] "Kernel 4.14 release," Web site. [Online]. Available: https://kernelnewbies.org/Linux_4.14

[16] "iproute2 4.14 release," Web site. [Online]. Available: https://mirrors.edge.kernel.org/pub/linux/utils/net/iproute2/

[17] "SERA - SEgment Routing Aware Firewall," Web site. [Online]. Available: https://github.com/SRouting/SERA

[18] "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," Web site. [Online]. Available: http://iperf.fr

[19] "iptables releases. iptables-1.6.2 changelog," Web site, February 2018. [Online]. Available: https://netfilter.org/projects/iptables/downloads.html#iptables-1.6.2

Charting the Complexity Landscape of Virtual Network Embeddings

Matthias Rost
Technische Universität Berlin
Email: mrost@inet.tu-berlin.de

Stefan Schmid
University of Vienna
Email: stefan_schmid@univie.ac.at

***Abstract*—Many resource allocation problems in the cloud can be described as a basic Virtual Network Embedding Problem (VNEP): the problem of finding a mapping of a *request graph* (describing a workload) onto a *substrate graph* (describing the physical infrastructure). Applications range from mapping testbeds (from where the problem originated), over the embedding of batch-processing workloads (virtual clusters) to the embedding of service function chains. The different applications come with their own specific requirements and constraints, including node mapping constraints, routing policies, and latency constraints. While the VNEP has been studied intensively over the last years, complexity results are only known for specific models and we lack a comprehensive understanding of its hardness.**

This paper charts the complexity landscape of the VNEP by providing a systematic analysis of the hardness of a wide range of VNEP variants, using a unifying and rigorous proof framework. In particular, we show that the problem of finding a feasible embedding is $\mathcal{NP}$-complete in general, and, hence, the VNEP cannot be approximated *under any objective*, unless $\mathcal{P}=\mathcal{NP}$ holds. Importantly, we derive $\mathcal{NP}$-completeness results also for finding approximate embeddings, which may violate, e.g., capacity constraints by certain factors. Lastly, we prove that our results still pertain when restricting the request graphs to planar or degree-bounded graphs.

I. Introduction

At the heart of the cloud computing paradigm lies the idea of efficient resource sharing: due to virtualization, multiple workloads can co-habit and use a given resource infrastructure simultaneously. Indeed, cloud computing introduces great flexibilities in terms of *where* workloads can be mapped. At the same time, exploiting this mapping flexibility poses a fundamental algorithmic challenge. In particular, in order to provide predictable performance, guarantees on all, i.e. node and edge, resources need to be ensured. Indeed, it has been shown that cloud application performance can suffer significantly from interference on the communication network [1].

The underlying algorithmic problem is essentially a graph theoretical one: both the workload as well as the infrastructure can be modeled as *graphs*. The former, the so-called *request graph*, describes the resource requirements both on the nodes (e.g., the virtual machines) as well as on the interconnecting network. The latter, the so-called *substrate graph*, describes the physical infrastructure and its resources (servers and links). Figure 1 depicts an example of embedding a request graph.

The problem is known in the networking community under the name *Virtual Network Embedding Problem* (VNEP) and has been studied intensively for over a decade [2], [3]. Besides the rather general study of the VNEP, which emerged originally from the study of testbed provisioning, essentially the same problems are considered in the context of Service Function Chaining [4], [5], as well as in the context of embeddings Virtual Clusters, a specific batch processing request abstraction [6], [7].

ISBN 978-3-903176-08-9

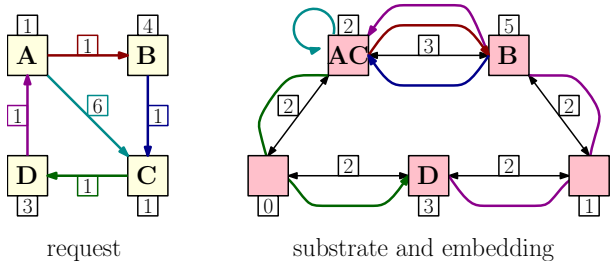


Fig. 1. Example request (left) together with an exemplary embedding on a substrate network (right). The numeric labels at the network elements denote the resource demands (of the request) and the available capacity (of the substrate), respectively. In the embedding, the single request edge $(\mathbf{C},\mathbf{D})$ (green) is realized via a path of length two in the substrate. As request nodes **A** and **C** are collocated on the same substrate node, the edge $(\mathbf{A},\mathbf{C})$ does not use any substrate edges and hence uses no edge resources. Note that the allocations induced by the embedding do not exceed the substrate's capacities.

A. Related Work

a) Objectives & Restrictions: Depending on the setting, many different objectives are considered for the VNEP. The most studied ones concern minimizing the (resource allocation) cost [2], [3], maximizing the profit by exerting admission control [8], [9], and minimizing the maximal load [4], [10].

Besides commonly enforcing that the substrate's physical capacities on servers and edges are not exceeded to provide Quality-of-Service [3], additional restrictions have emerged:

- Restrictions on the placement of virtual nodes first arose to enforce closeness to locations of interest [2], but were also used in the context of privacy policies to restrict mappings to certain countries [11]. However, these restrictions are now also used in the context of Service Function Chaining, as specific functions may only be mapped on x86 servers, while firewall appliances cannot [4], [5].
- Routing restrictions first arose in the context of expressing security policies, as for example some traffic may not be

TABLE I
OVERVIEW ON RESULTS OBTAINED IN THIS PAPER.

VNEP variants	⟨VE | -⟩	⟨E | N⟩	⟨V | R⟩	⟨- | NR⟩	⟨- | NL⟩
Enforcing Node Capacities	✓	⋆	✓	⋆	⋆
Enforcing Edge Capacities	✓	✓	⋆	⋆	⋆
Enforcing Node Placement Restrictions	⋆	✓	⋆	✓	✓
Enforcing Edge Routing Restrictions	⋆	⋆	✓	✓	⋆
Enforcing Latency Restrictions	⋆	⋆	⋆	⋆	✓

Results							
Section IV	$\mathcal{NP}$-completeness and inapproximability under any objective	Thm. 19	Thm. 20	Thm. 21	Thm. 22	Thm. 22
Section V	$\mathcal{NP}$-completeness and inapproximability when increasing *node capacities* by a factor $\alpha < 2$	Thm. 23	-	Thm. 23	-	-
	Inapproximability when increasing edge capacities by a factor $\beta \in \Theta(\log n / \log\log n)$ (unless $\mathcal{NP} \subseteq \mathcal{BP}$-*TIME*$(\bigcup_{d\geq 1} n^{d \log\log n})$)	Thm. 25	Thm. 25	-	-	-
	$\mathcal{NP}$-completeness and inapproximability when loosening latency bounds by a factor $\gamma < 2$	-	-	-	-	Thm. 24
Section VI	Results are preserved for acyclic substrates (except for Thm. 25)	Obs. 26				
	Results are preserved for acyclic, planar, degree-bounded requests	Thm. 31				

routed via insecure domains or physical links shall not be shared with other virtual networks [3], [12].

- Restrictions on latencies were studied for the VNEP in [13] and have been recently studied intensely in the context of Service Function Chaining to achieve responsiveness and Quality-of-Service [4], [5].

b) Algorithmic Approaches: Several dozens of algorithms were proposed to solve the VNEP and its siblings, including the Virtual Cluster Embedding [6] and Service Function Chain Embedding problem [3]. Most approaches to solve the VNEP either rely on heuristics [2] or metaheuristics [3]. On the other hand, several works study exact (non-polynomial time) algorithms to solve the problem to (near-)optimality or to devise heuristics. Mixed Integer Programming is the most widely used exact approach [4], [9], [13].

Only recently, approximation algorithms providing quality guarantees for the VNEP have been presented. In particular, the embedding of chains is approximated under assumptions on the requested resources and the achievable benefit in [14]. In [15] approximations for cactus request graphs are detailed, while [16] presents fixed-parameter tractable approximations for *arbitrary* request graph topologies.

c) Complexity Results: Surprisingly, despite the relevance of the problem and the large body of literature, the complexity of the underlying problems has not received much attention. While it can be easily seen that the Virtual Network Embedding Problem encompasses several $\mathcal{NP}$-hard problems as e.g. the k-disjoint paths problem [17], the minimum linear arrangment problem [18], or the subgraph isomorphism problem [19], most works on the VNEP cite a $\mathcal{NP}$-hardness result contained in a technical report from 2002 by Andersen [20]. The only other work studying the computational complexity is one by Amaldi et al. [21], which proved the $\mathcal{NP}$-hardness and inapproximability of the profit maximization objective while not taking into account latency or routing restrictions and not considering the hardness of embedding a *single* request.

Bibliographic Note: An extended version of this paper was published as a technical report [22]. In addition to the contents found in this paper, it contains a detailed Integer Programming formulation able to solve the VNEP variants considered in this paper. Furthermore, it contains the proof of Theorem 25, that is omitted here due to space constraints.

B. Contributions and Overview

In this work, we initiate the systematic study of the computational complexity of the VNEP. Taking all the aforementioned restrictions into account, we first compile a concise taxonomy of the VNEP variants in Section II. Then, we present a powerful reduction framework in Section III, which is the base for all hardness results presented in this paper. In particular, we show the following (see also Table I):

- We show the $\mathcal{NP}$-completeness of *five* different VNEP variants in Section IV. For example, we consider the variant only enforcing capacity constraints, but also one in which only node placement and latency restrictions must be obeyed *in the absence of capacity constraints*.
- We extend these results in Section V and show that the considered variants remain $\mathcal{NP}$-complete even when computing *approximate* embeddings, which may exceed latency or capacity constraints by certain factors.
- Lastly, we show in Section VI that the respective VNEP variants remain $\mathcal{NP}$-complete even when restricting substrate graphs to directed acyclic graphs (DAGs) and request graphs to planar, degree-bounded DAGs.

As we are proving $\mathcal{NP}$-completeness throughout this paper, the implications of our results are severe. Given the $\mathcal{NP}$-completeness of finding *any* feasible solution, finding an optimal solution *subject to any objective* is at least $\mathcal{NP}$-hard. Furthermore, unless $\mathcal{P} = \mathcal{NP}$ holds, the respective variants cannot be approximated to within *any* factor.

Table I summarizes our results and is to be read as follows. Any of the five rightmost columns represents a specific VNEP variant. The ✓ symbol indicates restrictions that are enforced, while the ⋆ symbol indicates restrictions which are not considered. Importantly, enabling a ⋆ restriction, does not change the results (cf. Lemma 9). Considering a specific variant, the respective column should be read from top to bottom. Considering for example ⟨ **VE** | **-** ⟩, its $\mathcal{NP}$-completeness is shown in Theorem 19 while its inapproximability when relaxing node capacity constraints is shown in Theorem 23. Lastly, all results also hold under the graph restrictions of the two bottom rows.

II. Formal Model

We now formally introduce the VNEP and its variants.

Notation: The following notation is used throughout this work. We use $[x]$ to denote $\{1, 2, \ldots, x\}$ for $x \in \mathbb{N}$. For a directed graph $G = (V, E)$, we denote by $\delta^+(v) \subseteq E$ and $\delta^-(v) \subseteq E$ the outgoing and incoming edges of node $v \in V$. When considering functions on tuples, we omit the parantheses of the tuple and simply write $f(a, b)$ instead of $f((a, b))$.

A. Basic Problem Definition

We refer to the physical network as substrate network and model it as directed graph $G_S = (V_S, E_S)$. Capacities in the substrate are given by the function $c_S : V_S \cup E_S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. The capacity $c_S(u)$ of node $u \in V_S$ may represent for example the number of CPUs while the capacity $c_S(u, v)$ of edge $(u, v) \in E_S$ represents the available bandwidth. By allowing to set substrate capacities to ∞, the capacity constraints on the respective substrate elements can be effectively disabled. We denote by $\mathcal{P}_S$ the set of all simple paths in G_S.

A request is similarly modeled as directed graph $G_r = (V_r, E_r)$ together with node and edge capacities (demands) $c_r : V_r \cup E_r \to \mathbb{R}_{\geq 0}$.

The task is to find a *mapping* of request graph G_r on the substrate network G_S, i.e. a mapping of request nodes to substrate nodes and a mapping of request edges to paths in the substrate. Virtual nodes and edges can only be mapped on substrate nodes and edges of sufficient capacity. Accordingly, we denote by $V_S^i = \{u \in V_S | c_S(u) \geq c_r(i)\}$ the set of substrate nodes supporting the mapping of node $i \in V_r$ and by $E_S^{i,j} = \{(u, v) \in E_S | c_S(u, v) \geq c_r(i, j)\}$ the substrate edges supporting the mapping of virtual edge $(i, j) \in E_r$.

Definition 1 (Valid Mapping)**.** A *valid* mapping of request G_r to the substrate G_S is a tuple $m = (m_V, m_E)$ of functions that map nodes and edges, respectively, s.t. the following holds:

- The function $m_V : V_r \to V_S$ maps virtual nodes to *suitable* substrate nodes, such that $m_V(i) \in V_S^i$ holds for $i \in V_r$.
- The function $m_E : E_r \to \mathcal{P}_S$ maps virtual edges $(i, j) \in E_r$ to simple paths in G_S connecting $m_V(i)$ to $m_V(j)$, such that $m_E(i, j) \subseteq E_S^{i,j}$ holds for $(i, j) \in E_r$. □

Considering the above definition, note the following. Firstly, the mapping $m_E(i, j)$ of the virtual edge $(i, j) \in E_r$ may be empty, if (and only if) i and j are mapped on the same substrate node. Secondly, the definition only enforces that single resource allocations do not exceed the available capacity. To enforce that the cumulative allocations respect capacities, we introduce the following:

Definition 2 (Allocations)**.** We denote by $A_m(x) \in \mathbb{R}_{\geq 0}$ the resource allocations induced by valid mapping $m = (m_V, m_E)$ on substrate element $x \in G_S$ and define

$$A_m(u) = \sum_{i \in V_r : m_V(i) = u} c_r(i)$$
$$A_m(u, v) = \sum_{(i,j) \in E_r : (u,v) \in m_E(i,j)} c_r(i, j)$$

for node $u \in V_S$ and edge $(u, v) \in E_S$, respectively. □

We call a mapping *feasible*, if the (cumulative) allocations do not exceed the capacity of any substrate element:

Definition 3 (Feasible Embedding)**.** A mapping m represents a feasible embedding, if the allocations do not exceed the capacity, i.e. $A_m(x) \leq c_S(x)$ holds for $x \in G_S$. □

In this paper we study the *decision* variant of the VNEP, asking whether there exists a feasible embedding:

Definition 4 (VNEP, Decision Variant)**.** Given is a single request G_r that shall be embedded on the substrate graph G_S. The task is to find a feasible embedding or to decide that no feasible embedding exists. □

B. Variants of the VNEP & Nomenclature

As discussed when reviewing the related work in Section I-A, additional requirements are enforced in many settings. Accordingly, we now formalize (i) node placement, (ii) edge routing, and (iii) latency restrictions. Node placement and edge routing restrictions effectively exclude potential mapping options for nodes and edges. For latency restrictions we introduce latency bounds for each of the virtual edges.

Definition 5 (Node Placement Restrictions)**.** For each virtual node $i \in V_r$ a set of forbidden substrate nodes $\overline{V}_S^i \subset V_S$ is provided. Accordingly, the set of allowed nodes V_S^i is defined to be $\{u \in V_S \setminus \overline{V}_S^i \,|\, c_S(u) \geq c_r(i)\}$. □

Definition 6 (Routing Restrictions)**.** For each virtual edge $(i, j) \in E_r$ a set of forbidden substrate edges $\overline{E}_S^{i,j} \subseteq E_S$ is provided. Accordingly, the set of allowed edges $E_S^{i,j}$ is set to be $\{(u, v) \in E_S \setminus \overline{E}_S^{i,j} \,|\, c_S(u, v) \geq c_r(i, j)\}$. □

Definition 7 (Latency Restrictions)**.** For each substrate edge $e \in E_S$ the edge's latency is given via $l_S(e) \in \mathbb{R}_{\geq 0}$. Latency bounds for virtual edges are specified via the function $l_r : E_r \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, such that the latency along the substrate path $m_E(i, j)$, used to realize the edge $(i, j) \in E_r$, is less than $l_r(i, j)$. Formally, the definition of feasible embeddings (cf. Definition 3) is extended by including that $\sum_{e \in m_E(i,j)} l_S(e) \leq l_r(i, j)$ holds for $(i, j) \in E_r$. □

We introduce the following taxonomy to denote the different problem variants.

Definition 8 (Taxonomy)**.** We use the notation $\langle \mathbf{C} \,|\, \mathbf{A} \rangle$ to indicate whether and which of the capacity constraints **C** and which of the additional constraints **A** are enforced.

C We denote by **V** node capacities, by **E** edge capacities, and by **-** that none are used. When node or edge capacities are *not* considered, we assume the capacities of the respective substrate elements to be ∞.

A Considering the additional restrictions, the abbreviations **-**, **N**, **L**, and **R** stand for no restrictions, node placement, latency, and routing restrictions, respectively. □

Accordingly, $\langle \mathbf{VE} \,|\, \mathbf{-} \rangle$ indicates the classic VNEP without additional constraints while obeying capacities and $\langle \mathbf{-} \,|\, \mathbf{NL} \rangle$ indicates the combination of node placement and latency

restrictions while neither considering node or edge capacities. We note that the introduction of more restrictions only makes the respective problem harder:

Lemma 9. *A VNEP variant* $\langle \mathbf{A} \,|\, \mathbf{C} \rangle$ *that encompasses all restrictions of* $\langle \mathbf{A'} \,|\, \mathbf{C'} \rangle$ *is at least as hard as* $\langle \mathbf{A'} \,|\, \mathbf{C'} \rangle$.

Proof. The capacity constraints as well as the additional requirements are all formulated in such a fashion that any one of these can be disabled. Considering capacities and latencies, one may set the respective substrate capacities to ∞ and the latencies of substrate edges to 0, respectively. For node placement and edge restrictions one may set the forbidden node and edge sets to the empty set. Hence, a trivial reduction from $\langle \mathbf{A} \,|\, \mathbf{C} \rangle$ to $\langle \mathbf{A'} \,|\, \mathbf{C'} \rangle$ exists and the result follows. ■

C. Relaxing Constraints: Approximate Embeddings

Within this work, we show the VNEP to be $\mathcal{NP}$-complete under many meaningful restriction combinations. This in turn also implies the inapproximability of the respective VNEP variants (unless $\mathcal{P} = \mathcal{NP}$ holds). Hence, it is natural to consider a broader class of (approximation) algorithms that may violate constraints by a certain factor: instead of answering the question whether a valid embedding exists that satisfies all capacity constraints, one might for example seek an embedding that uses at most two times the actual capacities. We refer to these embeddings as approximate embeddings:

Definition 10 (α- / β- / γ-Approximate Embeddings)**.** A mapping m is an approximate embedding, if it is valid and violates capacity or latency constraints only within a certain bound. Specifically, we call an embedding α- and β-approximate, when node and edge allocations are bounded by α and β times the respective node or edge capacity. Considering latency restrictions, we call a mapping γ-approximate when latencies are within a factor of γ of the original bound. Formally, the following must hold for $\alpha, \beta, \gamma \geq 1$:

$$\begin{aligned} A_m(u) &\leq \alpha \cdot c_S(u) && \forall u \in V_S \\ A_m(u,v) &\leq \beta \cdot c_S(u,v) && \forall (u,v) \in E_S \\ \sum_{e \in m_E(i,j)} l_S(e) &\leq \gamma \cdot l_r(i,j) && \forall (i,j) \in E_r \end{aligned}$$ □

III. Reduction Framework

This section presents the main insight and contribution of our paper, namely a generic reduction framework that allows to derive hardness results by slightly tailoring the proof for the individual problem variants. Our reduction framework relies on 3-SAT and we first introduce some notation. Afterwards we continue by constructing a (partial) VNEP instance, whose solution will indicate whether the 3-SAT formula is satisfiable.

A. 3-SAT: Notation and Problem Statement

We denote by $\mathcal{L}_\phi = \{x_k\}_{k \in [N]}$ a set of $N \in \mathbb{N}$ literals and by $\mathcal{C}_\phi = \{\mathcal{C}_i\}_{i \in [M]}$ a set of $M \in \mathbb{N}$ clauses, in which literals may occur either positively or negated. The formula $\phi = \bigwedge_{\mathcal{C}_i \in \mathcal{C}_\phi} \mathcal{C}_i$ is a 3-SAT formula, iff. each clause $\mathcal{C}_i$ is the disjunction of at most 3 literals of $\mathcal{L}_\phi$. Denoting the truth values by F and T, 3-SAT asks to determine whether an assignment $\alpha : \mathcal{L}_\phi \to \{\mathrm{F}, \mathrm{T}\}$ exists, such that ϕ is satisfied. 3-SAT is one of Karp's 21 $\mathcal{NP}$-complete problems:

Theorem 11 (Karp [23])**.** *Deciding 3-*SAT *is $\mathcal{NP}$-complete.*

For reducing 3-SAT to VNEP, it is important that the clauses be ordered and we define the following:

Definition 12 (First Occurence of Literals)**.** We denote by $\mathcal{C} : \mathcal{L}_\phi \to [M]$ the function yielding the index of the clause in which a literal first occurs. Hence, if $\mathcal{C}(x_k) = i$, then x_k is contained in $\mathcal{C}_i$ while not contained in $\mathcal{C}_{i'}$ for $i' \in [i-1]$. □

As we are interested the satisfiability of a 3-SAT formula ϕ, we define the set of satisfying assignments *per clause*:

Definition 13 (Satisfying Assignments)**.** We denote by $\mathcal{A}_i = \{a_{i,m} : \mathcal{L}_i \to \{\mathrm{F}, \mathrm{T}\} \mid a_{i,m} \text{ satisfies } \mathcal{C}_i\}$ the set of all possible assignments of truth values to the literals $\mathcal{L}_i$ of $\mathcal{C}_i$ satisfying $\mathcal{C}_i$. Note that all elements of $\mathcal{A}_i$ are functions. □

Lastly, to abbreviate notation, we employ $\mathcal{L}_{i,j} = \mathcal{L}_i \cap \mathcal{L}_j$ to denote the intersection of the literal sets of $\mathcal{C}_i$ and $\mathcal{C}_j$.

B. General VNEP Instance Construction

For a given 3-SAT formula ϕ, we now construct a VNEP instance consisting of a substrate graph $G_{S(\phi)}$ and a request graph $G_{r(\phi)}$. The question whether the formula ϕ is satisfiable will eventually reduce to the question whether a feasible embedding of $G_{r(\phi)}$ on $G_{S(\phi)}$ exists. Figure 2 illustrates the construction described in the following.

Definition 14 (Substrate Graph $G_{S(\phi)}$)**.** For a given 3-SAT formula ϕ we define the substrate graph $G_{S(\phi)} = (G_{S(\phi)}, E_{S(\phi)})$ as follows. For each clause $\mathcal{C}_i \in \mathcal{C}_\phi$ and each potential assignment of truth values satisfying $\mathcal{C}_i$, a substrate node is constructed, i.e. we set $V_{S(\phi)} = \bigcup_{\mathcal{C}_i \in \mathcal{C}_\phi} \mathcal{A}_i$. We connect two substrate nodes $a_{i,m} \in V_{S(\phi)}$ and $a_{j,n} \in V_{S(\phi)}$, iff. a literal x_k is introduced in the clause $\mathcal{C}_i$ for the first time and is also used in clause $\mathcal{C}_j$, *and* $a_{i,m}$ and $a_{j,n}$ agree on the truth values of the literals contained in both clauses. Formally, we set:

$$E_{S(\phi)} = \left\{ (a_{i,m}, a_{j,n}) \;\middle|\; \begin{array}{l} \exists x_k \in \mathcal{L}_{i,j} \text{ with } \mathcal{C}(x_k) = i \text{ and} \\ a_{i,m}(x_l) = a_{j,n}(x_l) \text{ for } x_l \in \mathcal{L}_{i,j} \end{array} \right\}$$

Capacities etc. are introduced in the respective reductions. □

Definition 15 (Request Graph $r(\phi)$)**.** For a given 3-SAT formula ϕ we define the request graph $G_{r(\phi)} = (V_{r(\phi)}, E_{r(\phi)})$ as follows. For each clause $\mathcal{C}_i \in \mathcal{C}_\phi$ a node v_i is introduced, i.e. $V_{r(\phi)} = \{v_i \,|\, \mathcal{C}_i \in \mathcal{C}_\phi\}$. Matching the construction of the substrate graph $G_{S(\phi)}$, we introduce directed edges $(v_i, v_j) \in E_{r(\phi)}$ only if there exists a literal $x_k \in \mathcal{C}_i$ being introduced in $\mathcal{C}_i$ and being also used in the clause $\mathcal{C}_j$:

$$E_{r(\phi)} = \{(v_i, v_j) \,|\, \exists x_k \in \mathcal{L}_{i,j} \text{ with } \mathcal{C}(x_k) = i\}$$

Demands etc. are introduced in the respective reductions. □

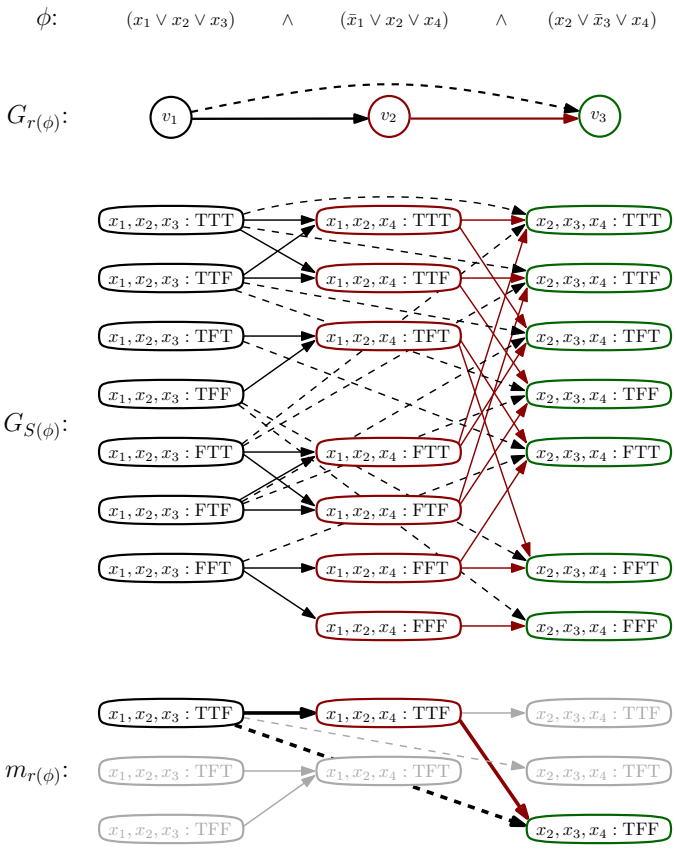


Fig. 2. Visualization of the construction of substrate and request graphs for the 3-SAT formula ϕ (cf. Definitions 14 and 15). Additionally, a mapping m satisfying the conditions of Lemma 16 is shown. Accordingly, the formula ϕ is satisfied. Concretely, the mapping represents the assignment of truth values $x_1 = \mathrm{T}$, $x_2 = \mathrm{T}$, $x_3 = \mathrm{F}$, $x_4 = \mathrm{F}$.

C. The Base Lemma

Nearly all of our results are based on the following lemma.

Lemma 16. *The 3-*SAT *formula ϕ is satisfiable*

if and only if

there exists a valid mapping m of $G_{r(\phi)}$ on $G_{S(\phi)}$, such that

1) *each virtual node v_i is mapped on a substrate node corresponding to assignments $\mathcal{A}_i$ of the i-th clause, i.e. $m_V(v_i) \in \mathcal{A}_i$ holds for all $v_i \in V_{r(\phi)}$, and*
2) *virtual edges are embedded using a single substrate edge, i.e. $|m_E(v_i, v_j)| = 1$ holds for all $(v_i, v_j) \in E_{r(\phi)}$.*

Proof. We first show that if ϕ is satisfiable, then such a mapping m must exist. Afterwards, we show that if such a mapping m exists, then ϕ must be satisfiable.

Assume that ϕ is satisfiable and let $\alpha : \mathcal{L}_\phi \to \{\mathrm{F}, \mathrm{T}\}$ denote an assignment of truth values, such that α satisfies ϕ. We construct a mapping $m = (m_V, m_E)$ for request $r(\phi)$ as follows. The virtual node $v_i \in V_{r(\phi)}$ corresponding to clause $\mathcal{C}_i$ is mapped onto the substrate node $a_{i,m} \in \mathcal{A}_i \subseteq V_{S(\phi)}$, iff. $a_{i,m}$ agrees with α on the assignment of truth values to the contained literals, i.e. $a_{i,m}(x_k) = \alpha(x_k)$ for $x_k \in \mathcal{C}_i$. As α satisfies ϕ, it satisfies each clause and hence $m_V(v_i) \in V_{S(\phi)}$ holds for all $\mathcal{C}_i \in \mathcal{C}_\phi$. The virtual edge $(v_i, v_j) \in E_{r(\phi)}$ is mapped via the direct edge between $m_V(v_i)$ and $m_V(v_j)$. This edge $(m_V(v_i), m_V(v_j))$ must exist in $E_{S(\phi)}$, as the existence of virtual edge (v_i, v_j) implies that clause $\mathcal{C}_i$ is the first clause introducing a literal of $\mathcal{L}_{i,j}$ and $m_V(v_i) = a_{i,m}$ and $m_V(v_j) = a_{j,n}$ must agree by construction on the assignment of truth values for all literals. Clearly, the constructed mapping m fulfills both the conditions stated in the lemma, hence completing the first half of the proof.

We now show that if there exists a mapping m meeting the two requirements stated in the lemma, then the formula ϕ is indeed satisfiable. We constructively recover an assignment of truth values $\alpha : \mathcal{L}_\phi \to \{\mathrm{F}, \mathrm{T}\}$ from the mapping m by iteratively extending the initially empty assignment. Concretely, we iterate over the mappings of the virtual nodes corresponding to clauses $\mathcal{C}_\phi$ one by one (according to the precedence relation of the indices). By our assumption on the node mapping, $m_V(v_i) \in \mathcal{A}_i$ holds. Accordingly, as the substrate node $m_V(v_i)$ represents an assignment of truth values to the literals of clause $\mathcal{C}_i$, we extend α by setting $\alpha(x_k) \triangleq \big[m_V(v_i)\big](x_k)$ for all literals x_k contained in $\mathcal{C}_i$.

We first show that this extension is always valid in the sense that previously assigned truth values are never changed. To this end, assume that the clauses $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{i-1}$ were handled without any such violations. Hence the literals $\bigcup_{j<i} \mathcal{L}_j$ have been assigned truth values in the first $i-1$ iterations not contradicting previous assignments. When extending α by the mapping of $m_V(v_i)$ in the i-th iteration, there are two cases to consider. First, if none of the literals $\mathcal{L}_i$ were previously assigned a truth value, i.e. $\mathcal{L}_i \cap \bigcup_{j<i} \mathcal{L}_j = \emptyset$ holds, then the extension of α as described above cannot lead to a contradiction. Otherwise, if $\mathcal{L}_{i,\mathrm{pre}} = \mathcal{L}_i \cap \bigcup_{j<i} \mathcal{L}_j \neq \emptyset$ holds, we show that extending α by $m_V(v_i) = a_{i,m}$ does not change the truth value of any literal x_k contained in $\mathcal{L}_{i,\mathrm{pre}}$.

For the sake of contradiction, assume that $x_k \in \mathcal{C}_i$ is a literal, for which $\alpha(x_k)$ does not equal $\big[m_V(v_i)\big](x_k)$. As x_k was previously assigned a value, there must exist a clause $\mathcal{C}_j$ in which x_k was first used, such that $j < i$ holds. Let $m_V(v_i) = a_{i,m} \in \mathcal{A}_i$ and $m_V(v_j) = a_{j,n} \in \mathcal{A}_j$. As the edge (v_j, v_i) is contained in $E_{r(\phi)}$ by definition and all edges are mapped using a single substrate edge by our assumptions, $m_E(v_i, v_j) = \langle (a_{j,n}, a_{i,m}) \rangle$ must hold. Hence, as $(a_{j,n}, a_{i,m}) \in E_{S(\phi)}$ must hold and edges are only introduced *if* assignments agree with each other, we have $\big[m_V(v_j)\big](x_k) = a_{j,n}(x_k) = a_{i,m}(x_k) = \big[m_V(v_i)\big](x_k)$. This contradicts our assumption that $\alpha(x_k) \neq \big[m_V(v_i)\big](x_k)$ holds. Hence, the extension of α is always valid.

By construction of the substrate graph $G_{S(\phi)}$, the node set $\mathcal{A}_i \subseteq V_{S(\phi)}$ contains only the assignments of truth values for the literals $\mathcal{L}_i$ of clause $\mathcal{C}_i \in \mathcal{C}_\phi$ that satisfy the respective clause. Hence, α satisfies all of the clauses and hence satisfies ϕ, completing the proof of the base lemma. ∎

The base lemma is the heart of our reduction framework for obtaining our results and we note that the construction of the substrate and the request graph is polynomial in the size of the 3-SAT formula. Indeed, the base lemma forms the basis for

polynomial-time reductions for the different VNEP decision variants. Concretely, consider some VNEP variant $\langle \mathbf{X} \,|\, \mathbf{Y} \rangle$. If this variant is 'expressive' enough such that any feasible embedding must meet the criteria of Lemma 16, then $\langle \mathbf{X} \,|\, \mathbf{Y} \rangle$ is – by reduction from 3-SAT – $\mathcal{NP}$-hard. Furthermore, the existence of an Integer Program for each of the VNEP variants (cf. technical report [22]) shows that the variants lie in $\mathcal{NP}$ and hence the successful application of the base lemma shows the $\mathcal{NP}$-completeness of the respective variants. As a result, for the considered VNEP variants, any optimization problem (e.g. cost) cannot be approximated within any factor. The following lemma formalizes this observation:

Lemma 17. *If there is a polynomial-time reduction from 3-SAT to the VNEP decision variant $\langle \mathbf{X} \,|\, \mathbf{Y} \rangle$, then the VNEP variant $\langle \mathbf{X} \,|\, \mathbf{Y} \rangle$ is $\mathcal{NP}$-complete. Furthermore, any optimization problem over the same set of constraints is (i) $\mathcal{NP}$-hard and (ii) inapproximable (within any factor), unless $\mathcal{P} = \mathcal{NP}$ holds.*

Lastly, the following lemma will prove useful when applying the base lemma.

Lemma 18. *Exactly one of the following two following properties holds for formula φ:*

1) *The clauses of ϕ can be ordered such that within the corresponding request graph $G_{r(\phi)}$ only the node $v_1 \in V_{r(\phi)}$ has no incoming edges.*
2) *φ can be decomposed into formulas φ_1 and φ_2, such that the sets of literals occurring in φ_1 and φ_2 are disjoint, while $\varphi = \varphi_1 \wedge \varphi_2$ holds. Hence, φ is satisfiable iff. φ_1 and φ_2 are (independently) satisfiable.*

Proof. We prove the statement by a greedy construction and assume that the clauses are initially unordered. We iteratively assign an index to the clauses, keeping track of which clauses were not assigned an index yet. Initially, pick any of the clauses and assign it the index 1. Now, iteratively choose any clause which contains a literal that already occurs in the set of indexed clauses. If no such clause exists, then the clauses already indexed and the clauses not indexed obviously represent a partition of the literal set and hence the second statement holds true. However, if the greedy step succeeded every time, then the following holds with respect to the constructed ordering: any virtual node v_i corresponding to clause $\mathcal{C}_i$, for $i > 1$, must have an incoming edge by Definition 15 as the clause overlapped with the already introduced literals. ■

IV. $\mathcal{NP}$-COMPLETENESS OF THE VNEP

We employ our reduction framework outlined in the previous section to derive a series of hardness results for the VNEP. In particular, we first show the $\mathcal{NP}$-completeness of the original VNEP variant $\langle \mathbf{VE} \,|\, \mathbf{-} \rangle$ in the absence of additional restrictions. Given this result, we investigate several other problem settings and show, among others, that also deciding $\langle \mathbf{-} \,|\, \mathbf{LN} \rangle$ is $\mathcal{NP}$-complete. Hence, even when the physical network does not impose any resource constraints (i.e., nodes and edges have infinite capacities), finding an embedding satisfying latency and node placement restrictions is $\mathcal{NP}$-complete. Again, it must be noted that adding further restrictions only renders the VNEP harder (cf. Lemma 9).

A. $\mathcal{NP}$-Completeness under Capacity Constraints

We first consider the most basic VNEP variant $\langle \mathbf{VE} \,|\, \mathbf{-} \rangle$.

Theorem 19. VNEP $\langle \mathbf{VE} \,|\, \mathbf{-} \rangle$ *is $\mathcal{NP}$-complete and cannot be approximated under any objective (unless $\mathcal{P} = \mathcal{NP}$).*

Proof. We show the statement via a polynomial-time reduction from 3-SAT according to Lemmas 16 and 17. Given is a 3-SAT formula ϕ. We assume for now that the first statement of Lemma 18 holds, i.e. that within the request graph $G_{r(\phi)}$ only the first node $v_1 \in V_{r(\phi)}$ has no incoming edge.

To enforce the properties of Lemma 16, we set the substrate and request capacities for some small λ, $0 < \lambda < 1/|\mathcal{C}_\phi|$, as follows. The capacity of substrate nodes is determined by the clause whose assignments they represent. Furthermore, the capacities decrease monotonically with each clause. Similarly, but now increasing per clause, the capacities of edges are determined by the clause that the edge's head corresponds to:

$$\begin{aligned} c_S(a_{i,m}) &= 1 + \lambda \cdot (M - i) \quad \forall \mathcal{C}_i \in \mathcal{C}_\phi, a_{i,m} \in \mathcal{A}_i \\ c_S(e) &= 1 + \lambda \cdot i \qquad\qquad \forall \mathcal{C}_i \in \mathcal{C}_\phi, e \in \delta^-(\mathcal{A}_i) \end{aligned}$$

The demands are set to match the respective capacities:

$$\begin{aligned} c_{r(\phi)}(v_i) &= 1 + \lambda \cdot (M - i) \quad \forall v_i \in V_{r(\phi)} \\ c_{r(\phi)}(e) &= 1 + \lambda \cdot i \qquad\qquad \forall v_j \in V_{r(\phi)}, e \in \delta^-(v_j) \end{aligned}$$

Due to the decreasing node demands and capacities, virtual node $v_j \in V_{r(\phi)}$ corresponding to clause $\mathcal{C}_i$ can only be mapped on substrate nodes $\bigcup_{k=1}^{j} \mathcal{A}_k$. Due to the choice of λ, the capacity of any substrate node is less than 2 while each virtual node has a demand larger than 1. Hence, two virtual nodes can never be collocated (mapped) on the same substrate node. Thus, all virtual edges must be mapped onto at least a single substrate edge. Considering the virtual edge $e = (v_i, v_j) \in E_{r(\phi)}$ with demand $c_{r(\phi)}(e) = 1 + \lambda \cdot j$, the virtual node v_j must be mapped on a substrate node having an incoming edge of at least capacity $1 + \lambda \cdot j$. As the edge capacities increase with the clause index, only the substrate nodes in $\bigcup_{k=j}^{M} \mathcal{A}_k$ satisfy this condition. Hence, if node v_j has an incoming edge, it can only be mapped on nodes in $\bigcup_{k=1}^{j} \mathcal{A}_k \cap \bigcup_{k=j}^{M} \mathcal{A}_k = \mathcal{A}_j$. As we assumed that the first statement of Lemma 18 holds for ϕ and hence all nodes $v_2, \ldots, v_M$ have an incoming edge, we obtain that the virtual node v_i must be mapped on $\mathcal{A}_i \subseteq V_{S(\phi)}$ for $i = 2, \ldots, M$. Considering the first node v_1, we observe that only nodes in $\mathcal{A}_1$ offer sufficient capacity to host v_1. Hence, any feasible embedding will obey the first statement of Lemma 16 regarding the node mappings.

We now show that *any* feasible mapping will also obey the second property of Lemma 16, namely, that any virtual edge is mapped on exactly one substrate edge. To this end, assume for the sake of contradiction that $(v_i, v_j) \in E_{r(\phi)}$ is not mapped on a single substrate edge. As v_i must be mapped

on some node $a_{i,m} \in \mathcal{A}_i$ and v_j must be mapped on some node $a_{j,n} \in \mathcal{A}_j$, and as both the request and the substrate are directed acyclic graphs, the mapping of edge (v_i, v_j) must route through at least one intermediate node. Denote by $a_{k,l} \in \mathcal{A}_k$ for $i < k < j$ the first intermediate node via which the edge (v_i, v_j) is routed. By construction, the capacity of the substrate edge $(a_{i,m}, a_{k,l})$ is $1 + \lambda \cdot k$. However, as $k < j$ holds and the edge (v_i, v_j) has a demand of $1 + \lambda \cdot j$, the edge (v_i, v_j) cannot be routed via $a_{k,l}$. Hence, the only feasible edges for embedding the respective virtual edges are the direct connections between any two substrate nodes.

Therefore, all *feasible* solutions indicate the satisfiability of the formula ϕ. Any algorithm computing a feasible solution to the VNEP obeying node and edge capacities, decides 3-SAT.

Lastly, we argue for the validity of our assumption on the structure of ϕ, namely that the first statement of Lemma 18 holds. If this were not to hold, then the second statement of Lemma 18 holds true and the formula can be decomposed (potentially multiple times) into disjoint subformulas $\varphi_1, \ldots, \varphi_k$, such that (i) $\varphi = \bigwedge_{i=1}^{k} \varphi_i$ holds, and (ii) such that the first condition of Lemma 18 holds for each subformula. Accordingly, assuming that an algorithm exists which can construct feasible embeddings whenever they exist, this algorithm can be used to decide the satisfiability of each subformula, hence deciding the original satisfiability problem. ■

B. $\mathcal{NP}$-completeness under Additional Constraints

Building on the above $\mathcal{NP}$-completeness proof, we can adapt it easily to other settings.

Theorem 20. VNEP $\langle \mathbf{E} \,|\, \mathbf{N} \rangle$ *is $\mathcal{NP}$-complete and cannot be approximated under any objective (unless $\mathcal{P} = \mathcal{NP}$).*

Proof. In this setting node placement restrictions and substrate edge capacities are enforced. We apply the same construction as in the proof of Theorem 19. Employing the node placement restrictions, we can force the mapping of virtual node $v_i \in V_{r(\phi)}$ onto substrate nodes $\mathcal{A}_i$ by setting $\overline{V}_S^{v_i} = V_{S(\phi)} \setminus \mathcal{A}_i$ for all $v_i \in V_{r(\phi)}$. By the same argument as before, virtual edges cannot be mapped onto paths as the intermediate nodes do not support the respective demand. ■

Theorem 21. VNEP $\langle \mathbf{V} \,|\, \mathbf{R} \rangle$ *is $\mathcal{NP}$-complete and cannot be approximated under any objective (unless $\mathcal{P} = \mathcal{NP}$).*

Proof. In this setting only node capacities must be obeyed, while routing restrictions may be introduced. We employ the same node capacities as in the proof of Theorem 19, such that virtual node $v_i \in V_{r(\phi)}$ may only be mapped on nodes $\bigcup_{k=1}^{i} \mathcal{A}_k$. Routing restrictions are set to only allow direct edges, i.e. $\overline{E}_S^{v_i,v_j} = E_{S(\phi)} \setminus (\mathcal{A}_i \times \mathcal{A}_j)$ holds for each $(v_i, v_j) \in E_{r(\phi)}$. Again, $v_1 \in V_{r(\phi)}$ must be mapped on a node in $\mathcal{A}_1$, while all other virtual nodes have at least one incoming edge according to Lemma 18. As multiple virtual nodes cannot be placed on the same substrate node and virtual edges must span at least one substrate edge, any node v_j can only be mapped on nodes in $\mathcal{A}_j$ for $j \in \{2, \ldots, M\}$. Together with the routing restrictions both requirements of Lemma 16 are safeguarded and the result follows. ■

Theorem 22. VNEP *variants* $\langle \text{-} \,|\, \mathbf{NR} \rangle$ *and* $\langle \text{-} \,|\, \mathbf{NL} \rangle$ *are $\mathcal{NP}$-complete and cannot be approximated under any objective (unless $\mathcal{P} = \mathcal{NP}$).*

Proof. In both cases capacities are not considered at all. Allowing for node placement restrictions, the first property of Lemma 16 is easily safeguarded (cf. proof of Theorem 20). By employing the same routing restrictions as in the proof of Theorem 21 the result follows directly for the case $\langle \text{-} \,|\, \mathbf{NR} \rangle$. Latency restrictions can be easily used to enforce that virtual edges do not span more than a single substrate edge. Concretely, we set unit substrate edge latencies and unit virtual edge latency bounds: if an edge was to be embedded via more than one edge, the latency restrictions would be violated. Hence, the result also holds for $\langle \text{-} \,|\, \mathbf{NL} \rangle$. ■

V. $\mathcal{NP}$-completeness of Computing Approximate Embeddings

Given the hardness results presented in Section IV, the question arises to which extent the hardness can be overcome when only computing approximate embeddings (cf. Definition 10), i.e. embeddings that may violate capacity constraints or exceed latency constraints by certain factors. Based on the proofs presented in Section IV, we first derive hardness results for computing α-approximate embeddings (allowing node capacity violations) and γ-approximate embeddings (allowing latency violations). For β-approximate embeddings (allowing edge capacity violations) a reduction from an edge-disjoint paths problem is presented in our technical report [22].

Theorem 23. *For* $\langle \mathbf{VE} \,|\, \text{-} \rangle$ *and* $\langle \mathbf{V} \,|\, \mathbf{R} \rangle$ *finding an α-approximate embedding is $\mathcal{NP}$-complete as well as inapproximable under any objective (unless $\mathcal{P} = \mathcal{NP}$) for any $\alpha < 2$.*

Proof. Assume that there exists an algorithm computing α-approximate embeddings for $\alpha = 2 - \varepsilon$, $0 < \varepsilon < 1$. We adapt the proofs of Theorem 19 and 21 slightly. First, note that for α-approximate mappings validity still has to hold according to Definition 10. Hence, by the decreasing node capacities the virtual node v_j can only be mapped on substrate nodes $\bigcup_{k=1}^{j} \mathcal{A}_k$. Furthermore, by either enforcing edge capacities or edge routing restrictions, the node v_j can still only be mapped on $\mathcal{A}_j$. Hence, the only missing piece to show that the respective proofs still hold is the fact that still at most a single virtual node can be mapped on a single substrate node. To ensure, that this still holds, we adapt the capacities. Concretely, we choose λ, such that $\lambda < \varepsilon/(2 \cdot |\mathcal{C}_\phi|)$ holds. Hence, the capacity of any substrate node is less than $1 + \varepsilon/2$. By relaxing the capacity constraints by the factor $2 - \varepsilon$, the allowed substrate node allocations are upper bounded by $(1 + \varepsilon/2) \cdot (2 - \varepsilon) = 2 - \varepsilon - \varepsilon^2/2 < 2$. As the demand of any virtual node is larger than 1, still at most a single virtual node can be mapped on a substrate node. Hence, the respective proofs still apply and the results follow. ■

The result for γ-approximate embeddings can be obtained in a very similar fashion.

Theorem 24. *For $\langle$ - $|$ NL $\rangle$ finding an γ-approximate embedding is $\mathcal{NP}$-complete as well as inapproximable under any objective (unless $\mathcal{P}=\mathcal{NP}$) for any $\gamma < 2$.*

Proof. The proof of Theorem 22 relied on the fact that due to the latency constraints each virtual edge must be mapped on a single substrate edge. As the latencies of substrate edges are uniformly set to 1 and all latency bounds are 1 as well, computing a γ-approximate embedding for $\gamma < 2$ implies that each virtual edge can still only be mapped on a single substrate edge. Hence, the result of Theorem 22 remains valid. ■

For β-approximate embeddings similar results can be obtained via a reduction from an edge-disjoint paths problem. Due to space constraints we only state our main result and refer the reader to our technical report for the proof [22].

Theorem 25. *Finding a β-approximate embedding for the* VNEP *variants $\langle$ VE $|$ - $\rangle$ and $\langle$ E $|$ N $\rangle$ is hard to approximate for $\beta \in \Theta(\log n / \log\log n)$, $n = |V_S|$, unless $\mathcal{NP} \subseteq \mathcal{BP}$-TIME$(\bigcup_{d\geq 1} n^{d \log\log n})$ holds.*

Note that above $\mathcal{BP}$-TIME$(f(n))$ denotes the class of problems solvable by probabilistic Turing machines in time $f(n)$ with bounded error-probability [24].

VI. $\mathcal{NP}$-completeness under Graph Restrictions

All of our $\mathcal{NP}$-completeness results (except Theorem 25) are based on a reduction from 3-SAT, yielding a specific directed-acylic substrate graph $G_{S(\phi)}$ and a specific directed acyclic request graph $G_{r(\phi)}$ and we note the following.

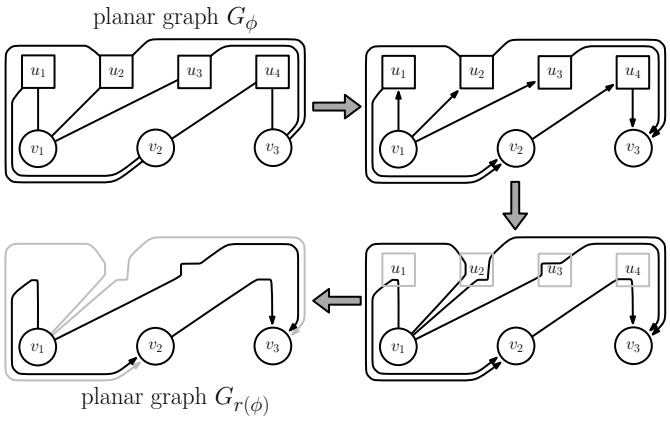


Fig. 3. Depicted is the transformation process of a planar graph G_ϕ (cf. Definition 27) to the planar graph $G_{r(\phi)}$. Concretely, the example formula of Figure 2 is revisited, i.e. $\phi = C_1 \wedge C_2 \wedge C_3$, with $C_1 = x_1 \vee x_2 \vee x_3$, $C_2 = \bar{x}_1 \vee x_2 \vee x_4$, and $C_3 = x_2 \vee \bar{x}_3 \vee x_4$. In the first step all edges are directed, such that edges from clause nodes are oriented towards literal nodes iff. the literal occurs in the respective clause for the first time (according to the ordering of clause nodes). In the second step, each outgoing edge of a literal node is joined with the single incoming edge (duplicating it when necessary), hence allowing to remove the literal nodes. In the last step, duplicate edges are removed, yielding the request graph $G_{r(\phi)}$. Each step of this transformation process safeguards the graph's planarity.

Observation 26. Theorems 19 - 24 still hold when restricting the request and the substrate to acyclic graphs.

Given the hardness of the VNEP and as for example Virtual Clusters (an undirected star network) can be optimally embedded in polynomial time [7], one might ask whether the hardness is preserved when restricting request graphs further.

In this section, we derive the result that the VNEP variants considered in this paper remain $\mathcal{NP}$-complete when request graphs are planar and degree-bounded. Our results are obtained by considering a *planar* variant of 3-SAT. The planarity of a formula ϕ is defined according ϕ's graph interpretation:

Definition 27 (Graph G_ϕ of formula ϕ)**.** The graph $G_\phi = (V_\phi, E_\phi)$ of a SAT formula ϕ is defined as follows. V_ϕ contains a node v_i for each clause $C_i \in \mathcal{C}_\phi$ and a node u_k for each literal $x_k \in \mathcal{L}_\phi$. An undirected edge $\{v_i, u_k\}$ is contained in E_ϕ, iff. the literal x_k is contained in C_i (either positive or negative). Note that the graph G_ϕ is bipartite. □

An example for the interpretation G_ϕ is depicted in Figure 3. Kratochvíl [25] considered the following variant of 4P3C-3-SAT and proved its $\mathcal{NP}$-completeness.

Definition 28 (4P3C-3-SAT)**.** The 4-Bounded Planar 3-Connected 3-SAT (4P3C-3-SAT) considers only 3-SAT formulas ϕ for which the following holds:
(1) In each clause, exactly 3 distinct literals are used.
(2) Each literal occurs in at most 4 clauses.
(3) The graph G_ϕ is planar.
(4) The graph G_ϕ is vertex 3-connected. □

Theorem 29 ([25])**.** 4P3C-3-SAT *is $\mathcal{NP}$-complete.*

The following lemma connects 4P3C-3-SAT formulas ϕ with the corresponding request graphs $G_{r(\phi)}$.

Lemma 30. *Given a* 4P3C-3-SAT *formula ϕ, the following holds for the request graph $G_{r(\phi)}$ (cf. Definition 15):*
1) The request graph $G_{r(\phi)}$ is planar.
2) The node-degree of $G_{r(\phi)}$ is bounded by 12.

Proof. We consider an arbitrary 4P3C-3-SAT formula ϕ to which the conditions of Definition 28 apply. We first show that the corresponding request graph $G_{r(\phi)}$ is planar by detailing a transformation process leading from G_ϕ to $G_{r(\phi)}$ while preserving planarity (see Figure 3 for an illustration).

Starting with the undirected graph G_ϕ, the edges are first oriented: an edge is oriented from a clause node to a literal node iff. the literal occurs in the respective clause for the first time according to the clauses' ordering. Note that while many reductions in Section IV required the reordering of clause nodes according to Lemma 18, this reordering preserves planarity as the structure of the graph G_ϕ does not change.

Given this directed graph, the literal nodes are now removed by joining the single incoming edge of the literal nodes with *each* outgoing edge of the corresponding literal node. In particular, consider the literal node x_2 of Figure 3: the single incoming edge (C_1, x_2) is joined with the outgoing edges (x_2, C_2) and (x_2, C_3) to obtain the edges (C_1, C_2)

and (C_1, C_3), respectively. As the duplication of the single incoming edge cannot refute planarity and all incoming and outgoing edges connect to the same node, the planarity of the graph is preserved in this step. Lastly, duplicate edges are removed to obtain the graph $G_{r(\phi)}$, which is, in turn, planar.

It remains to show, that the request graph $G_{r(\phi)}$ corresponding to ϕ exhibits a bounded node-degree of 12 (in the undirected interpretation of the graph $G_{r(\phi)}$). To see this, we note the following: based on the first two conditions of Definition 28, each clause node connects to exactly 3 literal nodes and each literal node connects to at most 4 clause nodes. Hence, when removing the literal nodes in the transformation process, the degree of each node may increase at most by a factor of 4. As any clause node had 3 neighboring literal nodes, this implies that the degree of any node is at most 12 after the transformation process, completing the proof. ■

Given the above, we easily derive the following theorem:

Theorem 31. *Theorems 19 - 24 hold when restricting the request graphs to be planar and / or degree 12-bounded. Theorem 25 holds for planar and degree 1-bounded graphs.*

Proof. Our $\mathcal{NP}$-completeness proofs in Section IV and Section V (except for Theorem 25) relied solely on the reduction from 3-SAT to VNEP using the base Lemma 16. As formulas of 4P3C-3-SAT are a strict subset of the 3-SAT formulas, the base Lemma 16 is still applicable for 4P3C-3-SAT formulas. However, due to the structure of 4P3C-3-SAT formulas, the corresponding requests in the reductions are planar and exhibit a node-degree bound of 12 by Lemma 30. Hence, solving the VNEP is $\mathcal{NP}$-complete, even when restricting the requests to planar and / or degree-bounded ones. Lastly, as proven in our technical report [22], Theorem 25 holds for planar and degree 1-bounded requests, concluding the proof. ■

VII. Conclusion

We presented a comprehensive set of hardness results for the VNEP and its variants, which lie at the core of many resource allocation problems in networks. Our results are negative in nature: we show that the problem variants are $\mathcal{NP}$-complete and hence inapproximable (unless $\mathcal{P} = \mathcal{NP}$) and that this holds true even for restricted classes of request graphs.

We believe that our results are of great importance for future work on several of the virtual network embedding problems. For example, our results on the variant enforcing node placement and latency restrictions are of specific interest for Service Function Chaining. Surprisingly, the respective problem is hard even when not considering any capacity constraints. Furthermore, we have shown that it is hard to compute embeddings satisfying latency bounds within a factor of (less than) two times the original bounds. In turn, whenever latency bounds must be obeyed strictly, one needs to rely on exact algorithmic techniques as e.g. Integer Programming.

Acknowledgements

This work was partially supported by Aalborg University's PreLytics project as well as by the German BMBF Software Campus grant 01IS1205.

References

[1] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 5, 2012.

[2] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM*, 2009.

[3] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, 2013.

[4] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014.

[5] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, Oct. 2015.

[6] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, New York, NY, USA, 2011.

[7] M. Rost, C. Fuerst, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, 2015.

[8] G. Even, M. Medina, G. Schaffrath, and S. Schmid, "Competitive and deterministic embeddings of virtual networks," *Theoretical Computer Science*, vol. 496.

[9] M. Rost, S. Schmid, and A. Feldmann, "It's about time: On optimal virtual network embeddings under temporal flexibilities," in *IEEE 28th International Parallel and Distributed Processing Symposium*, 2014.

[10] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, Feb. 2012.

[11] G. Schaffrath, S. Schmid, and A. Feldmann, "Optimizing long-lived cloudnets with migrations," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 2012.

[12] L. R. Bays, R. R. Oliveira, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Security-aware optimal resource allocation for virtual network embedding," in *Proceedings of the 8th International Conference on Network and Service Management*. IFIP, 2013.

[13] J. Inführ and G. R. Raidl, "Introducing the virtual network mapping problem with delay, routing and location constraints," in *Network Optimization*, J. Pahl, T. Reiners, and S. Voß, Eds. Springer, 2011.

[14] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in sdns," in *Structural Information and Communication Complexity*, J. Suomela, Ed. Springer, 2016.

[15] M. Rost and S. Schmid, "Virtual Network Embedding Approximations: Leveraging Randomized Rounding," in *(to appear) Proceedings IFIP Networking 2018*, 2018, preprint available at arXiv:1803.03622 [cs.NI]. [Online]. Available: http://arxiv.org/abs/1803.03622.

[16] ——, "(FPT-)Approximation Algorithms for the Virtual Network Embedding Problem," Tech. Rep. arXiv:1803.04452 [cs.NI], March 2018. [Online]. Available: http://arxiv.org/abs/1803.04452

[17] J. Chuzhoy, V. Guruswami, S. Khanna, and K. Talwar, "Hardness of routing with congestion in directed graphs," in *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, 2007.

[18] J. Díaz, J. Petit, and M. Serna, "A survey of graph layout problems," *ACM Computing Surveys*, vol. 34, no. 3, Sep. 2002.

[19] D. Eppstein, "Subgraph isomorphism in planar graphs and related problems," in *Graph Algorithms and Applications I*, 2011.

[20] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, [Online]. Available: http://repository.cmu.edu/compsci/86/.

[21] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213 – 220, 2016.

[22] M. Rost and S. Schmid, "NP-Completeness and Inapproximability of the Virtual Network Embedding and Its Variants," *CoRR*, vol. abs/1801.03162, 2018. [Online]. Available: http://arxiv.org/abs/1801.03162

[23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[24] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[25] J. Kratochvíl, "A special planar satisfiability problem and a consequence of its np-completeness," *Discrete Applied Mathematics*, vol. 52, no. 3, 1994.

Breaking Service Function Chains with Khaleesi

Sara Ayoubi, Shihabur Rahman Chowdhury, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
{sayoubi | sr2chowdhury | rboutaba}@uwaterloo.ca

***Abstract*—Network Function Virtualization (NFV) has recently emerged as a means to replace vendor specific, purpose built equipment with commodity hardware and leverage the open APIs and application orchestration for on demand deployment and scaling of network services. A well studied problem in NFV is the orchestration of Service Function Chains, (SFCs), *i.e.*, a set of Virtual Network Functions (VNFs) chained together to realize a network service. State-of-the-art literature on SFC orchestration assumes a strict traversal order of VNFs in an SFC and less attention has been paid to SFCs with relaxed VNF orderings. In this paper, we address the problem of *Flexible Service Function Chain Orchestration* that jointly allocates compute and network resources for SFCs while considering a relaxed traversal order for some pairs of VNFs. We propose *Khaleesi*, a suite of solutions that consists of: (i) an Integer Linear Program (ILP) for optimally solving the problem; and (ii) a heuristic algorithm to scale to larger instances of the problem. Our simulation results show that flexible SFCs can increase revenue earned per unit cost by as much as ≈10% compared to a rigid SFC.**

I. Introduction

Network Function Virtualization (NFV) [1] has received significant traction in recent years for its ability to decouple packet processing software, *i.e.*, *Network Functions* (NFs) from specialized hardware *middleboxes*. NFV proposes to run the NFs as *Virtual Network Functions (VNFs)* on commodity hardware and leverages advances in application orchestration for on-demand provisioning of Service Function Chains (SFCs), *i.e.*, an ordered sequence of VNFs that implements a network service. The capability of on-demand service provisioning and the consolidation of multiple NFs on commodity hardware enable the network operators to reduce their operational and capital expenditures.

However, the full benefits offered by NFV cannot be fully reaped without efficient resource allocation mechanisms for provisioning the SFCs. As a result, a significant body of research has been dedicated to address the problem of SFC orchestration, *i.e.*, joint allocation of compute and network resources to provision SFCs with Quality of Service (QoS) (*e.g.*, minimum bandwidth or maximum delay) requirements. SFC orchestration has been shown to be NP-hard [2] and a number of its variants have been studied in the research literature [3]. Despite the significant research efforts in solving SFC orchestration, little attention has been paid to the actual semantics of the VNFs while allocating resources for them. As a result, SFC has been mostly considered as a rigid sequence of VNFs, *i.e.*, the order of the VNFs cannot be modified. In this work, we take a closer look at the semantics of VNFs and show that VNF traversal order can be changed without modifying the semantics of an SFC, and that such flexibility can be leveraged to perform better resource allocation for SFCs.

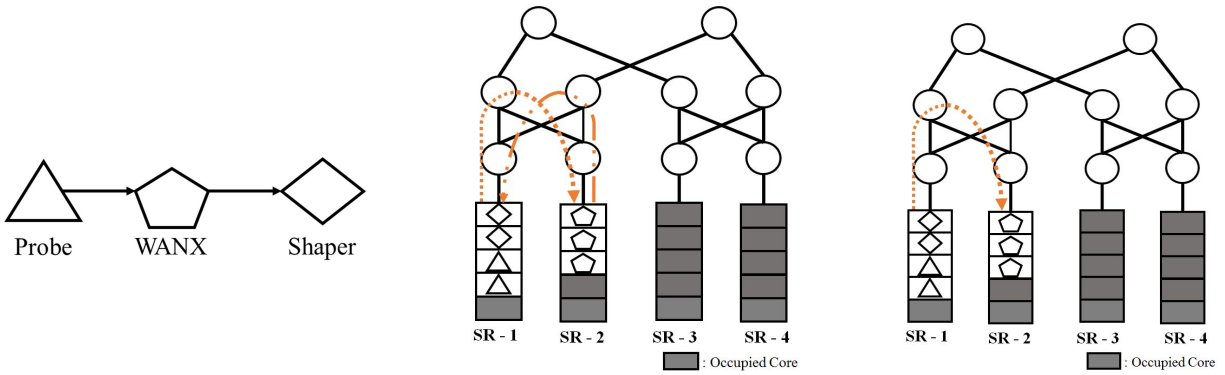
As an illustrative example, consider the following VNFs: a WAN optimizer that compresses and decompresses HTTP payload and a Probe that counts flows with a given layer 3 and layer 4 signature. These VNFs work on disjoint parts of a packet. Therefore, if they are next to each other in an SFC, swapping their order will neither affect the set of packets exiting the chain, nor the internal state of the VNFs (we call such VNFs *reorder-compatible*). Now consider the example SFC in Fig. 1(a), where the Probe, WAN Optimizer and Shaper requires 2, 3, and 2 CPU cores, respectively. If we consider the SFC to be rigid (*i.e.*, the order of VNFs cannot be modified) then the only possible provisioning solution is the one shown in Fig. 1(b). However, if we consider the reorder-compatibility of Probe and the WAN Optimizer, then we can swap their order in the SFC and provision the SFC as shown in Fig. 1(c). Note that, by leveraging the flexibility in VNF ordering, we provisioned the same SFC with 50% less network bandwidth.

Considerations for flexible SFCs are not entirely new. Early works in this area [4]–[6] proposed languages and data models to represent flexible SFC requests. However they do not discuss how the flexibility can be determined in the first place. More recently, Parabox [7] and NFP [8] proposed to parallelize VNF execution in an SFC by introducing additional components. In contrast, we do not assume any additional components for changing the order of VNFs in an SFC. Moreover, no quantifiable results exist in the research literature that demonstrates if any benefit can be gained from flexibility in VNF ordering. In this paper, we fill this gap in research literature with the following contributions:

- Theoretical foundation for determining reorder compatibility of VNFs and the mathematical models to represent such flexibility in an SFC.
- The first quantifiable result showing the benefits of flexible SFC orchestration over its rigid counterpart. Our empirical results demonstrate as much as 10% improvement in revenue earned per unit cost compared to rigid SFCs.
- *Khaleesi*[1], a suite of solutions to the Flexible SFC orchestration problem consisting of: (i) *OPT-Khaleesi*, an Integer Linear Program (ILP) formulation for optimally solving the flexible SFC orchestration problem. To the best of our knowledge, this is the first optimal solution proposed for such problem, and (ii) *FAST-Khaleesi*, A

[1]A character in popular fantasy novel *"A Song of Ice and Fire"*, who is also known as *the breaker of chains*

ISBN 978-3-903176-08-9 2018 IFIP



(a) Service Function Chain

(b) Placement of a Rigid SFC

(c) Flexible SFC (Probe and WANX swapped)

Fig. 1. Motivational Example

heuristic algorithm to solve larger instances of the problem. Simulation results show that *FAST-Khaleesi* allocates $\approx 2\times$ extra resources and accepts $\approx 20\%$ less SFC requests on average compared to the optimal solution.

The rest of the paper is organized as follows. We begin with a discussion of related work in Section II. Next, we present the theoretical foundations for identifying re-order compatible VNFs in Section III. In Section IV we present the system model and formally define the problem. We present our ILP formulation in Section V, followed by the heuristic in Section VI. Our evaluation results are presented in Section VII. In Section VIII we provide a brief discussion on the operational aspects that may limit the flexibility of SFCs. Finally, we conclude with some future research directions in Section IX.

II. Related Work

Since the publication of introductory white paper in late 2013, research in NFV has gained significant traction over the past few years. In the following, we discuss the state-of-the-art in SFC orchestration with a specific focus on research that considers relaxed order of VNFs in an SFC.

SFC orchestration is one of the most well studied problem in NFV. It has been addressed with different objective functions *e.g.*, maximizing the number of admitted SFCs [9], minimizing operational cost [2], minimizing the number of servers used [10], minimizing network resource utilization [11], minimizing the number of VNF instances used [4], [12] among others. For a comprehensive survey on resource allocation in NFV readers are referred to [3]. Approaches to solve these problems include variants of Linear Programming [4], [10], [11], Cut-and-solve method [9], polynomial time heuristic design [2], [12], approximation algorithm [13], *etc*. However, majority of these works assume a rigid SFC and do not leverage the flexibility of reorder compatibility between NFs.

Only a handful of research has considered relaxed ordering of NFs in an SFC. For instance [5] proposes a context free grammar to represent an SFC request with flexible parts. The flexible parts are segments of NFs that can be traversed in any arbitrary order. An extension of this work is presented in [6] where the authors propose a YANG data model to represent flexible structures. The authors also propose a Pareto-optimal solution and a heuristic to allocate resources for such SFCs with flexible parts. However, works such as [5], [6], [14] neither quantify the advantages in resource allocation stemming from flexible VNF ordering in an SFC over its rigid counterpart, nor do they shed light on the aspects that may affect such flexibility. More recently, Parabox [7] and NFP [8] proposed to relax the strict ordering of VNFs in an SFC and parallelize some of them to reduce end-to-end latency. However, to do so, additional components were introduced for splitting incoming packets to parallelized functions and also to combine their output. This incurs overhead in terms of network resources and processing delays. In contrast, we identify "re-order compatible" VNFs that can be swapped without adding any additional functions.

III. Re-order Compatibility of VNFs

Flexible SFCs can bring benefits in terms of resource allocation, thereby freeing up resources for more SFCs to be admitted. This can indeed increase infrastructure provider's revenue in the long run. Flexible SFCs have been preliminarily addressed before in [6] and [7]. However, in [6], the flexibility of the chain is an input to the orchestration problem. In [7], [8], the authors identify *independent* VNFs to increase parallelism in the chain. The concept of independent VNFs is somewhat similar to reorder compatible VNFs, however, not every independent pair of VNFs is re-order compatible. To the best of our knowledge, no existing work has yet formally defined re-order compatible VNFs. In the following, we lay the necessary theoretical foundation for identifying reorder compatible VNFs.

In a nutshell, two VNFs are considered re-order compatible if swapping their order in an SFC does not violate the SFC's semantics, *i.e.*, results in two *semantically equivalent* SFCs. We formally define semantically equivalent SFCs as follows:

Definition 1. ***Semantically Equivalent SFC:*** *Two SFCs S_1 and S_2 composed of the same set of VNFs, $\mathcal{F}$, in different order are semantically equivalent if: (i) for an ordered sequence of input packets p_{in}, both S_1 and S_2 produce identical ordered sequence of output packets, and (ii) after processing a packet*

$p \in p_{in}$ *the internal state of any VNF* $f_i \in \mathcal{F}$ *is identical in both* S_1 *and* S_2.

As a packet traverses an SFC, a VNF in the SFC performs any combination of the following three actions: (i) reads from the packet, (ii) modifies the packet, and (iii) updates its own internal state. For instance, while traversing a NAT, the source IP and source MAC address of a packet are modified, as well as the NAT's address translation table. In another instance, a probe may keep a count of the number of UDP packets that are sent/received and updates the count after a packet passes through it. Since SFCs provide a form of value-added service, therefore it is important to ensure that any flexible structure of the SFC provides the same service to the flows as well as the constituent VNFs have identical internal states.

To formalize this, we refer to the set of packet[2] fields that a VNF reads or modifies as "interest fields", denoted as $\mathcal{H}_f^i$, and the set of packet fields that affects the internal state of a VNF as "state fields", denoted as $\mathcal{H}_f^s$. Every $h \in \mathcal{H} = \mathcal{H}_f^i \cup \mathcal{H}_f^s$ is expressed as a (byte_offset, byte_length) pair (*e.g.*, source MAC can be expressed as a pair (6, 6)), which allows us to express interest fields and header fields in a protocol agnostic way (similar to [15]). By comparing the interest and state fields of two VNFs, we can determine their re-order compatibility.

Two VNFs are re-order compatible when their interest and state fields are mutually exclusive (*e.g.*, an application-layer firewall and a network-layer firewall). Furthermore, even when two VNFs share the same interest and/or state fields, as long as they do not modify the shared interest and/or state fields their processing functions remain independent (*e.g.*, a probe and a Deep Packet Inspector (DPI)). For each VNF, we represent the set of interest and state fields by a $|\mathcal{H}|\times$ 3 binary matrix $\mathcal{M}$, illustrated in Table I. The rows in $\mathcal{M}$ represent the different fields $h \in \mathcal{H}$. The columns r and w indicate whether this particular VNF, f, reads and/or modifies h, respectively. The column x indicates whether h affects the internal state of f, denoted by x = if($h \in \mathcal{H}_f^s$); $x \in \{0,1\}$. Given two VNFs u and v and their corresponding matrices $\mathcal{M}_u$ and $\mathcal{M}_v$, respectively, u and v are re-order compatible if:

$$\begin{aligned} \forall h \in \mathcal{H}, \forall (k,k') \in \{(u,v),(v,u)\} : \\ (\mathcal{M}_k[h][r] \wedge \mathcal{M}_{k'}[h][w]) \vee \\ (\mathcal{M}_k[h][w] \wedge \mathcal{M}_{k'}[h][w]) \vee \\ (\mathcal{M}_k[h][x] \wedge \mathcal{M}_{k'}[h][r]) \;=\; 0 \end{aligned} \tag{1}$$

Finding the interest and state fields of a VNF is a non-trivial task and is in fact a separate problem on its own. Different approaches have been taken in the past including middlebox modeling [16] [17] [18], header space analysis [19], white-box testing [20], black-box testing [21], *etc*. However, it is an orthogonal problem and is out of the scope of this paper.

A. Illustrative Example

Table II illustrates a re-order compatibility matrix for some commonly deployed NFs [22], namely firewall, web proxy, Intrusion Prevention System (IPS), Traffic Shaper, NAT, payload-DPI, and WAN Optimizer (WANX). We obtained the interest and state fields for these different VNFs by investigating existing middlebox models [16], middlebox catalog [23], IETF drafts [24], Click configuration files [25], and related research literature [7]. By applying (1) on the obtained interest and state fields, we obtained the matrix in Table II.

[2]In the rest of the paper by packet we refer to a Layer-2 frame.

TABLE I
INTEREST & STATE FIELD MATRIX $\mathcal{M}$

	r	w	x
h_1			
h_2			
....			
$h_{\|\mathcal{H}\|}$			

TABLE II
REORDER COMPATIBILITY MATRIX

	Firewall	Proxy	IPS	Shaper	NAT	DPI	WANX	Probe
Firewall			✓	✓		✓	✓	
Proxy			✓	✓		✓		
IPS	✓	✓		✓				✓
Shaper	✓	✓	✓			✓	✓	
NAT							✓	
DPI	✓	✓		✓				✓
WANX	✓			✓	✓			✓
Probe			✓			✓	✓	

We observe that a network Firewall (besides changing the MAC address) typically examines layer 2-4 headers, *e.g.*, source IP, destination IP, and port numbers, and either forwards or drops a packet. An IPS analyzes the packet (header and/or payload) and takes automated actions (drops packet, blocks traffic, sends alarm, or resets connection). Since a network firewall and an IPS perform "read-only" actions on common interest fields they can be swapped without affecting chain semantics. A traffic shaper classifies network traffic for QoS, a payload-DPI inspects the packet payload and raises an alarm if the packet matches a malicious signature. These VNFs are clearly re-order compatible with a network Firewall. Finally, a WANX can perform functions such as payload compression/de-compression, QoS tagging *etc*., which do not affect the interest and state fields of a network firewall. This renders them also re-order compatible. However, a network firewall and a network probe may not be re-order compatible because of the Probe's internal state. For instance, if a probe is counting the number of incoming connections to port 80, placing a firewall before the probe will yield a different count than placing it after the probe.

IV. SYSTEM MODEL AND PROBLEM STATEMENT

We first present a mathematical representation of the inputs, *i.e.*, the substrate network and the SFC request, then we formally define the problem of orchestrating SFCs with flexible VNF ordering with lowest resource provisioning cost (Flexible SFC orchestration for short).

A. Substrate Network

We represent the substrate network (SN) as an undirected graph $G^s = (N, L)$; where every node $n \in N$ is associated

with residual compute resource capacity c_n and internal-switching capacity b_n. Further, every link $l : (i, j) \in L$ is associated with residual bandwidth capacity $b_{i,j}$.

B. Virtual Network Functions

The set of available VNFs is represented by $\mathcal{F}$. Each VNF $f \in \mathcal{F}$ has compute resource requirement (*e.g.*, number of CPU cores) d_f. Additionally, we have a $|\mathcal{F} \times \mathcal{F}|$ matrix $\mathcal{R}$ that represents reorder compatibility between the VNFs. $R_{f,f'} = 1$ if VNF $f \in \mathcal{F}$ and $f' \in \mathcal{F}$ are reorder compatible according to the definition in Section III, 0 otherwise.

C. SFC Request

We represent an SFC request as an directed graph $G^v = (F, E, n_o, n_t)$, where F is the set of VNFs in the SFC. Note that we focus on SFCs that are structured as chains. Considerations for branching in SFCs is left as a future work. The requested chain E is represented by a set of directed virtual links, where each virtual link $e \in E$ consists of a pair of consecutive VNFs, and is associated with bandwidth demand d_e. Further, each SFC is associated with an ingress node $n_o \in N$ and an egress node $n_t \in N$.

D. Problem Statement

Given an SN G^s, set of VNFs $\mathcal{F}$, a reorder compatibility matrix $\mathcal{R}$, and a SFC request G^v:

- Place VNF $f \in F$ on a server $n \in N$.
- Route exactly $|F| - 1$ virtual links between the VNFs to form a chain structure.
- Map each virtual link either to a single substrate path or to a server.
- The cost of allocating bandwidth from the network to route the virtual links are minimized subject to the following constraints:
  - Servers and network links cannot be over-committed to accommodate VNFs and the virtual links.
  - A virtual link should be routed on a single path in the network or placed inside a single server in the network.

Flexible SFC orchestration bares some similarity with the well studied Virtual Network Embedding (VNE) problem [26]. However, a fundamental difference between flexible SFC orchestration and VNE is that the set of virtual links to be embedded is given as an input to VNE. Whereas, in case of flexible SFC orchestration this set is not known and is part of the solution instead.

V. *OPT-Khaleesi*: ILP Formulation

In this section, we first showcase our solution approach that transforms an SFC to exploit flexibility, followed by OPT-*Khaleesi*, our ILP model that optimally solves the flexible SFC orchestration problem.

A. SFC Transformation

In order to fully exploit the flexibility in a given chain, the model must become aware of every re-order compatible pair of VNFs in this chain. To give the model a complete knowledge of this flexibility, we augment the chain with directed virtual links. The ensemble of the original chain and the augmented directed virtual links represent all possible chains that can be traced. The model then selects the optimal subset of $|F|$-1 virtual links that routes through every VNF in the chain with minimum cost. Given an SFC and a re-order compatibility matrix $\mathcal{R}$, we augment the set E into E' as follows:

- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \in E$, links $(u \rightarrow v.\text{next})$ and $(v \rightarrow u)$ are created and added to E'.
- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \notin E$, if $\mathcal{R}_{u,j}$ = 1 $\forall\ u.next \leq j \leq v.prev$, links $(u \rightarrow v.\text{next})$ and $(v \rightarrow u)$ are created and added to E'.
- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \notin E$, if $\mathcal{R}_{j,v}$ = 1 $\forall\ u.next \leq j \leq v.prev$, links $(u \rightarrow v)$ and $(v \rightarrow u)$ are created and added to E'.

Fig. 2(c) shows the set E' generated for the SFC presented in Fig. 2(a) given the re-order compatibility matrix in Fig. 2(b). Generating the set E' alone is not sufficient; this is because not every subset $\bar{E} \subset E'$ renders a valid chain. Concretely, consider again the chain presented in Fig. 2(c), and recall that VNFs 0 and 1 are not re-order compatible. Here, while chain $\{3{\rightarrow}0{\rightarrow}1{\rightarrow}2\}$ is valid, chain $\{1{\rightarrow}2{\rightarrow}3{\rightarrow}0\}$ is not. Note that both chains are made of a combination of virtual links in E'. Subsequently, we need to ensure that any selected chain structure $\bar{E} \subset E'$ does not violate any semantics. To do so, we introduce Ω, a binary matrix, that indicates whether VNF f can precede function f'.

$$\Omega_{f,f'} = \begin{cases} 1, \text{if } (f, f') \in E \vee (R_{f,f'} = 1 \wedge \text{if } (f, f') \in E'), \\ 0, \text{otherwise} \end{cases}$$

Theorem 1. *Any chain that adheres to Ω is semantically valid.*

Proof. Let $\bar{E} \subset E'$ be semantically invalid. This means that $\bar{E}$ contains a pair of VNFs (f,f') where f cannot precede f'. If f cannot precede f', it means that f and f' are not re-order compatible; thus $R_{f,f'}$ = 0 and by definition $\Omega_{f,f'}$ = ($R_{f,f'} \wedge$ if $(f, f') \in E') = 0$. This follows that any chain that adheres to Ω is semantically valid. □

B. Decision Variables

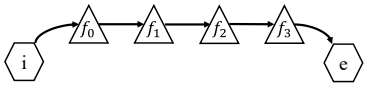
A VNF is placed on a node in the SN, which is represented by the following decision variable:

$$\theta_n^f = \begin{cases} 1 & \text{if VNF } f \in F \text{ is placed on node } n \in N, \\ 0 & \text{otherwise.} \end{cases}$$

The following determines the selection of a virtual link $e \in E'$ for inclusion in the final SFC:

$$z_e = \begin{cases} 1 & \text{if virtual link } e \in E' \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

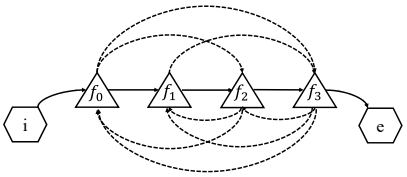
We use the binary variables x_n^e and y_n^e to indicate placement of the origin and destination of a virtual link $e \in E'$ on a



(a) Service Function Chain

	f_0	f_1	f_2	f_3
f_0	0	0	1	1
f_1	0	0	1	1
f_2	1	1	0	1
f_3	1	1	1	0

(b) Re-order Compatibility Matrix



(c) Augmented Set E'

Fig. 2. Semantically Correct Chains

substrate node $n \in N$. Routing of a virtual link $e \in E'$ is determined by the following variable:

$$w^e_{i,j} = \begin{cases} 1 & \text{if } e \in E' \text{ routed on substrate link } (i,j) \in L, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we use the following variable to derive the ordering between VNFs in the resultant SFC:

$$\delta_{f,f'} = \begin{cases} 1 & \text{if VNF } f \text{ precedes VNF } f' \text{ in the resultant SFC,} \\ 0 & \text{otherwise.} \end{cases}$$

C. Constraints

1) VNF Placement Constraints: Constraint (2) ensures that each VNF is placed on at most a single substrate node. Placement of the origin and destination of each virtual link is ensured by constraints (3) and (4), respectively. Finally, we ensure by (5) that either both the source and destination of a virtual link are placed, or neither.

$$\forall f \in F : \sum_{n \in N} \theta^f_n = 1 \tag{2}$$

$$\forall e \in E', n \in N : x^e_n \leq \sum_{f \in F : o(e) = f} \theta^f_n \tag{3}$$

$$\forall e \in E', n \in N : y^e_n \leq \sum_{f \in F : t(e) = f} \theta^f_n \tag{4}$$

$$\forall e \in E' : \sum_{n \in N} x^e_n - \sum_{n \in N} y^e_n = 0 \tag{5}$$

2) SFC Selection Constraints: The first and last VNF in the resultant SFC is determined by (6) and (7), respectively. Then, (8) ensures that a virtual link is routed *iff* both of its endpoints are placed. Constraints (9) and (10) ensure that every VNF in the resultant SFC is traversed exactly once. We ensure that exactly $|F| - 1$ virtual links are placed by (11). Finally, (12) is used to break loops between virtual links.

$$\forall \bar{e} \in E' : \{o(\bar{e}) = n_o\}, \forall f \in F : \sum_{n \in N} y^{\bar{e}}_n \leq 1 - \sum_{\substack{e \in E' : \{e \neq \bar{e}\}, \\ \{t(e) = f\}}} z_e \tag{6}$$

$$\forall \bar{e} \in E' : \{t(\bar{e}) = n_t\}, \forall f \in F : \sum_{n \in N} x^{\bar{e}}_n \leq 1 - \sum_{\substack{e \in E' : \{e \neq \bar{e}\}, \\ \{o(e) = f\}}} z_e \tag{7}$$

$$\forall e \in E' : z_e \leq \frac{1}{2}\Big(\sum_{n \in N} x^e_n + \sum_{n \in N} y^e_n\Big) \tag{8}$$

$$\forall f \in F : \sum_{e \in E' : (o(e) = f)} \sum_{n \in N} x^e_n \leq 1 \tag{9}$$

$$\forall f \in F : \sum_{e \in E' : (t(e) = f)} \sum_{n \in N} y^e_n \leq 1 \tag{10}$$

$$\sum_{e \in E'} z_e = |F| - 1 \tag{11}$$

$$\forall e \in E', e' \in E' : \{o(e) = t(e') \wedge t(e) = o(e')\} : z_e + z_{e'} \leq 1 \tag{12}$$

3) Substrate Capacity Constraints: (13), (14), and (15) represent the server capacity, internal switching capacity, and substrate link capacity constraints, respectively.

$$\forall n \in N : \sum_{f \in F} \theta^f_n \cdot d_f \leq c_n \tag{13}$$

$$\forall f \in F : \sum_{e \in E'} (x^e_n \cdot y^e_n) \cdot d_e \leq b_n \tag{14}$$

$$\forall (i,j) \in L : \sum_{e \in E'} \sum_{(i,j) \in L} w^e_{i,j} \cdot d_e \leq b_{i,j} \tag{15}$$

4) SFC Routing Constraints: Constraint (16) represents the flow conservation for mapping the virtual links. We use (17) to determine the ordering of VNFs in the resultant SFC. Finally, (18) ensures that the order of VNFs preserves the SFC's semantics.

$$\forall i \in N, e \in E' : \sum_{j : (i,j \in L)} w^e_{i,j} - \sum_{j : (j,i) \in L} w^e_{j,i} = x^e_i - y^e_i \tag{16}$$

$$\forall f \in F, \forall f' \in F, \forall f'' \in F : \delta_{f,f''} \geq (\delta_{f,f'} \cdot \delta_{f',f''}) + z_{e : \{o(e) = f, \\ t(e) = f''\}} \tag{17}$$

$$\forall f \in F, \forall f' \in F : \delta_{f,f'} \leq \Omega_{f,f'} \tag{18}$$

Note that (14) and (17) contain product of two integer variables, which renders the model non-linear. However, the product of two integer variables can be linearized as follows: For Constraint (14), we introduce a new variable $g^e_n \in \{0, 1\}$

such that:

$$g_n^e \leq x_n^e \tag{19}$$

$$g_n^e \leq y_n^e \tag{20}$$

$$g_n^e \geq x_n^e + y_n^e - 1 \tag{21}$$

Similarly, we linearize Constraint (17) by introducing a new variable $q_{f,f',f''} \in \{0,1\}$ such that

$$q_{f,f',f''} \leq \delta_{f,f'} \tag{22}$$

$$q_{f,f',f''} \leq \delta_{f',f''} \tag{23}$$

$$q_{f,f',f''} \geq \delta_{f,f'} + \delta_{f',f''} - 1 \tag{24}$$

D. Objective Function

Our objective is to embed the SFC while minimizing the incurred cost in terms of bandwidth consumption.

$$\text{minimize} \sum_{e \in E'} \sum_{(i,j) \in L} w_{i,j}^e$$

E. Hardness of the Problem

Theorem 2. *OPT-Khaleesi is NP-Complete.*

Proof. Given a graph G^s=(N,L), where c_n = 1 $\forall$ $n \in N$ and b_l = 1 $\forall$ $l \in L$. We transform G^s into G'^s=(N',L') by adding an auxiliary node of capacity 0 between every pair (i,j) $\in$ L. G'^s thus represents an SN where some nodes are servers and the rest are network nodes. Now assume that we have a *chaotic* SFC request of size N with 1 unit demand for each VNF in the SFC and d_f = 1. A chaotic SFC refers to an SFC where all pair of VNFs are re-order compatible. Solving the Hamiltonian path problem in G^s corresponds to finding a path that spans every node in N. This is exactly solving the flexible service chaining problem in G'^s since the Hamiltonian path will span N compute nodes in G^s (the size of the chain). Conversely, solving the flexible service chaining in G'^s would mean that we have found a chain that spans N compute nodes. This chain in G'^s corresponds to a Hamiltonian path in G^s. Since computing Hamiltonian path is NP-Complete, therefore a special case of our problem (*i.e.*, placement of a chaotic chain of size N) is also NP-Complete. Therefore, by restriction, *OPT-Khaleesi* is NP-complete. □

VI. *FAST-Khaleesi*: Heuristic Solution

To overcome the computational complexity of *OPT-Khaleesi*, we propose *FAST-Khaleesi*, a heuristic that performs flexible chaining, VNF placement, and routing with the objective to minimize bandwidth footprint. *FAST-Khaleesi* consists of 4 Steps, and is designed to select the chain that encourages more virtual links to be routed internally, which in turn reduces the number of inter-server switching.

- **Step 1: Generate all SFCs** First, given the set E', we trace all possible chains that do not violate any semantics (*i.e.*, respect Ω) by using backtracking. We denote this set as $\mathcal{S}$. Generating all possible chains for a SFC where all VNFs are re-order compatible may yield an exponential number of combinations $O(|F|!)$ in the worst case. However, in practice, the size of SFC does not exceed 6 in a DC [27]. Moreover, not all pairs of VNFs are typically reorder compatible with each other. Therefore, in practice the actual number of valid chains is far less than the worst case.
- **Step 2: Find Candidate Servers** Step 2 consists of finding a list of candidate servers. A candidate server is a server with sufficient CPU resources to accommodate any VNF. Once the list of candidate servers is obtained, we compute the shortest path between each candidate server $n \in \bar{N}$ and the ingress n_o and egress n_t nodes of the chain, respectively. We denote these distances as x_o and x_t, respectively. Subsequently, we compute the ratio r_n = $\frac{c_n}{x_o + x_e}$ for every node $n \in \bar{N}$. By sorting the servers in decreasing order of ratio r, we prioritize the servers with highest capacity and proximity to the chain's ingress and egress nodes. This will also potentially reduce the bandwidth footprint, as the VNFs will be placed close to the origin and sink of the chain at hand. The time complexity of Step 2 is O ($2 \cdot \bar{N} \cdot (L + N\log N) + \bar{N}\log\bar{N}) \leq$ O($N^2 \log N$) by dropping lower-order terms.
- **Step 3: VNF Placement** The VNF placement is performed for every chain as follows: First, for every candidate server $n \in \bar{N}$ and for every VNF $f \in F$, we

Algorithm 1: FAST-*Khaleesi* Algorithm

Input: G^s = (N,L), G^v = (F, E, n_o, n_t), E'
Output: NF Placement and Chain Routing Solution m

```
function FAST-Khaleesi
    Step 1: Generate all Valid Chains S
    Step 2: Find all candidate servers in N
    N̄ = {}
    forall n ∈ N do
        if (c_n < min_{∀f∈F} d_f) then
            continue
        N̄ = N̄ ∪ {n}
        r_n = |shortestPath(n, n_0)|+|shortestPath(n, n_t)|
    SortDescendingOrder(N̄, c/r)
    M = {}; /*Initialize an empty solution set*/
    forall s = (F, Ē) ∈ S do
        Step 3: VNF Placement
        m̄ = {} /*Initialize an empty solution*/
        V = F
        while (|V| > 0) do
            F̄ = FindMinOpCost_{∀v∈V}(s, N̄)
            m̄.nmap = m̄.nmap ∪ (F̄, n)
            V = V - F̄
            Ē = Ē - GetInternallySwitchedLinks(F̄)
        Step 4: Virtual Link Routing
        forall (e ∈ Ē) do
            m̄.emap = m̄.emap ∪ Dijkstra(e)
        M = M ∪ {m̄}
    m = FindLowestCostSol(M)
    return m
```

compute an opportunity cost of placing f along with a subset of VNFs $\bar{F}$ on n. The subset $\bar{F} \cup f$ with the lowest opportunity cost is chosen and mapped on node n. Here, opportunity cost refers to the number of virtual links that require inter-server switching as result of placing $\bar{F} \cup f$ on n. This cost is reduced by maximizing the number of virtual links with both end points placed on the same server. Hence, to minimize the opportunity cost for every VNF f, we traverse the chain s starting at f, and every time we find a pair of adjacent VNFs that can be packed in n without violating its capacity we add them to $\bar{F}$. This iteration is repeated until the placement of all VNFs is settled. At every iteration, the set of virtual links that require inter-server routing is updated. The time complexity of Step 3 is $O(|S| \cdot |\bar{N}| \cdot |F|^2)$.

- **Step 4: Inter-Server Routing** Finally, for every placement solution generated in Step 3, Dijkstra's shortest path algorithm is performed to route the set of virtual links whose end points are distributed in different servers. The time complexity of Step 4 is $O(|S| \cdot (|L| + |N| \log |N|))$.

Once Steps 3 and 4 are performed for every SFC, the algorithm terminates, and the mapping solution with the lowest total bandwidth consumption is returned. If no feasible mapping solution can be found for any chain, the SFC is rejected.

VII. Performance Evaluation

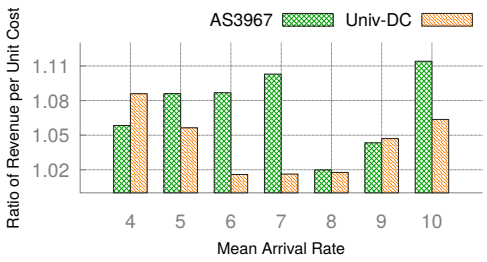
We evaluate our proposed solutions in the following scenarios: (i) evaluating the benefits of flexible VNF ordering in SFCs (Section VII-C), and (ii) performance comparison between *FAST-Khaleesi* and *OPT-Khaleesi* (Section VII-D). Before presenting the results, we describe our simulation setup in Section VII-A and the evaluation metrics in Section VII-B.

A. Simulation Setup

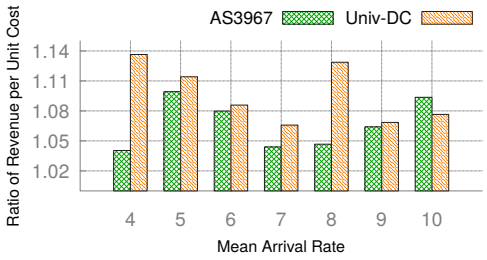
1) Testbed: We implemented *OPT-Khaleesi* and *FAST-Khaleesi* using IBM ILOG CPLEX 12.5 Java libraries and Java, respectively. Our testbed consists of machines with hyper-threaded Intel 8×10-core Xeon E7-8870 CPU and 1TB of memory. We developed an in-house discrete event simulator to simulate the arrival and departure of SFCs on an SN.

2) Topology: We used the following two real topologies, representing two different scenarios and scales for our evaluation: (i) Univ-DC: a university data center topology with 23 nodes and 42 links from [28]; and (ii) AS3967: a moderate size ISP topology with 79 nodes and 147 links from Rocketfuel dataset [29]. (i) represents a data center network, which typically has high server density compared to an ISP's backbone network such as (ii). To represent this diversity in server density we augmented (i) and (ii) with 144 and 64 servers, respectively, each with 8 CPU cores.

3) Traffic Data: For Univ-DC topology, we used real traffic traces from the same data center [28] to generate SFC request between pairs of edge switches. For AS3967, no real traffic traces are available, hence, we resorted to generating synthetic traffic. We used FNSS tool [30] and generated time varying traffic by following the distribution presented in [31].



(a) Orchestration of Flexible vs Rigid SFCs



(b) Impact of considering flexible VNF ordering in existing SFC orchestration algorithm [2]

Fig. 3. Benefits of Flexible VNF ordering in SFC

4) Middlebox Data: We selected a set of six VNFs from the ones listed in Table II and computed the reorder compatibility matrix accordingly. We randomly chained subsets of the selected VNFs to generate SFCs with lengths between 3 and 6. Middlebox CPU requirements were obtained from the research literature and available vendor data sheets [2], [32]. SFC arrival and departure was generated following a Poisson distribution with mean arrival rate varied between 4 - 10 SFCs per 100 time unit. Mean life time of these SFCs were set to 1000 time units. The simulation was run for a total of 10000 time units and included 400 - 960 SFC requests. Note that the dataset and parameters chosen for evaluation are in accordance with relevant research literature [3], [26].

B. Evaluation Metrics

1) Acceptance Ratio: The ratio of number of admitted SFC requests to the total number of SFC requests.

2) Embedding Path Length: Embedding path length is the sum of lengths of all the paths used for routing all inter-NF links in an SFC.

3) Revenue per unit cost: Revenue is computed as a function that is proportional to an SFCs total resource requirement. Revenue earned per unit cost is calculated by dividing revenue from an SFC by the SFC's embedding cost.

C. Benefits of Flexibility in SFCs

We demonstrate the benefits of flexible VNF ordering in SFCs by evaluating the following two scenarios. First, we compare results of optimally orchestrating flexible SFCs with that of orchestrating non-flexible or rigid SFCs. In the second scenario, we empirically evaluate how much benefit we can get by feeding all possible SFCs stemming from a flexible SFC to an existing orchestration algorithm from the literature.

The goal is to evaluate how much benefit we can get even without explicitly considering flexible VNF ordering in an existing SFC orchestration algorithm. For that purpose, we use the dynamic programming algorithm from [2].

For the first scenario, we implemented an ILP similar to [2] for comparison. From the results, we did not observe much of a difference between the two approaches in terms of acceptance ratio for different arrival rates. However, the number of accepted SFCs does not say much about the types of SFCs that were accepted. Therefore, we further analyzed the solutions and computed the revenue earned per unit cost for different arrival rates and present the result in Fig. 3(a). More specifically, we present the ratio of revenue earned per unit cost for flexible and rigid cases. The flexible case always yielded more revenue per unit embedding cost to an extent of 11% compared to the rigid cases.

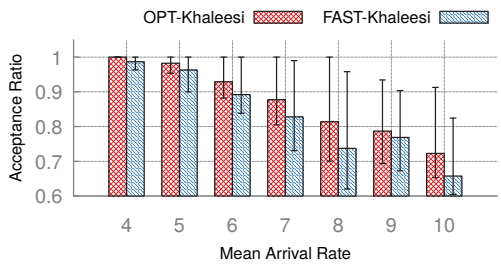
For the second scenario, we take a black box approach. Instead of modifying the dynamic programming algorithm from [2], we executed the algorithm for all valid SFCs that can be traced from an SFC request. We keep the result with the lowest cost for each SFC request. We compare the results from this setting with that from executing [2] on non-flexible SFCs. The results are presented in Fig. 3(b). We observe that even without explicitly considering and exploiting flexibility in SFCs, there is about 10% improvement in revenue earned per unit cost on average for [2].

The takeaway from this study is that flexibility in SFCs can yield more revenue per unit cost even for an SFC orchestration algorithm not designed to handle flexible SFCs. An intuition behind such result is that the flexibility in an SFC leads to reordering of some of the VNFs to accept some more resource demanding ones compared to the non-flexible case.

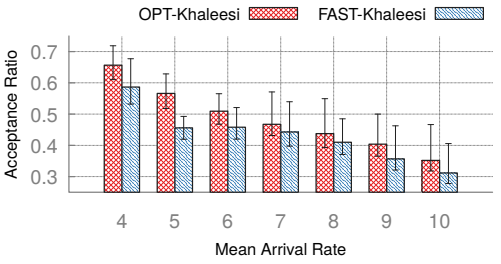
D. FAST-Khaleesi vs. OPT-Khaleesi

1) Acceptance Ratio: We present the cumulative acceptance ratio of *OPT-Khaleesi* and *FAST-Khaleesi* for both SN topologies in Fig. 4 with 5th and 95th percentile error bars. We consider the first 1000 time units of simulation as the warm up period and discard results from that period. For a compute resource constraint environment as in AS3967, the performance gap between the heuristic and the optimal is larger than Univ-DC topology. Nonetheless, we found the heuristic to accept no greater than 20% less SFC requests on average compared to the optimal solution.

2) Mean Embedding Path Length: Our cost function is proportional to the embedding path length for an SFC. Therefore, we compare the mean embedding path length across all SFC requests to gain an estimate as to how much far off is the heuristic from the optimal solution. The results for both of the topologies are presented in Fig. 5. For the data center topology where network diameter is relatively smaller compared to the ISP topology, the heuristic's mean embedding path was within 20% of that of the optimal solution. However, on a network with larger diameter such as the ISP topology we used, this stretch increased up to $\approx 2\times$ the optimal solution.

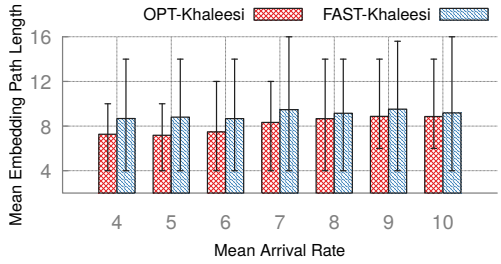


(a) Univ-DC Topology

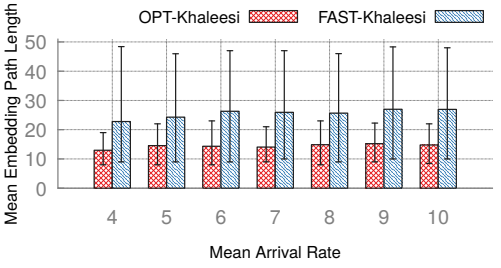


(b) AS3967 Topology

Fig. 4. Acceptance Ratio vs. Load



(a) Univ-DC Topology

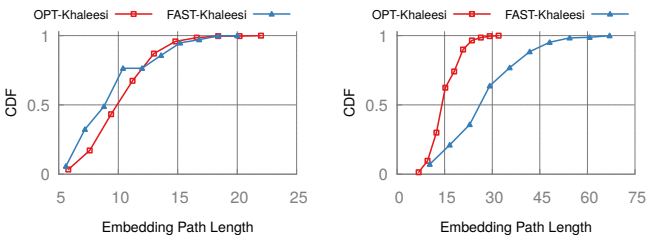


(b) AS3967 Topology

Fig. 5. Comparison of Mean Embedding Path Length

We also show the Cumulative Distribution Function (CDF) of embedding path length of one fixed arrival rate in Fig. 6. The purpose is to show the breakdown of the length distribution and try to understand the long errorbars in Fig. 5. As we can see, the heuristic demonstrates a long tailed CDF. This long tail is due to the heuristic's shortcoming of finding a good solution when the network is very close to saturation.

3) Execution Time: Execution time for *OPT-Khaleesi* and *FAST-Khaleesi* for embedding a single SFC request is presented in Table III. Note that execution time for Univ-DC is higher than that of AS3967. This is because, majority of the execution time in both of the solutions is spent to find



(a) Univ-DC Topology (b) AS3967 Topology

Fig. 6. Mean Embedding Path Length vs. Load

TABLE III
AVERAGE EXECUTION TIME PER SFC REQUEST

Topology	*OPT-Khaleesi*	*FAST-Khaleesi*
Univ-DC	4538ms	61ms
AS3967	1920ms	43ms

suitable placement of VNFs on the servers. Recall that the data center topology has higher number of servers, hence, higher execution time despite being smaller than AS3967.

VIII. DISCUSSION

In light of the above, we clearly can see potential advantages of flexible structures over the rigid ones. However, the degree of flexibility that might be available in a network not only depends on the types of VNFs deployed, but also on the operator's policies and customer requirements. There are certain NFs, which by policy might not be flexible at all. For instance, a network's policy or a customer's requirement may govern that all incoming and outgoing traffic must traverse through a firewall. In such cases, even if the VNF is reorder compatible with others, it cannot be considered for such flexibility. However, there are indeed some NFs that are deployed for performance enhancement purposes such as a WAN Optimizer, Video Transcoder, Traffic Shaper *etc*., that may have less strict requirement on their order. Our solution can also work with such operator policies. In such cases, where we have additional constraints imposed by policies, setting appropriate entries in $\mathcal{R}$ before running the optimizations should be sufficient.

IX. CONCLUSION

In this paper, we have taken a first step towards studying the impact of flexibility in SFCs and also how such flexibility can be leveraged for resource allocation. We present the first quantifiable results showing the potential benefits of flexible SFCs over rigid ones. Our results show that indeed there is improvement in revenue per unit cost, however, the significance of this improvement can be better explained when it is translated into actual monetary values. To the best of our knowledge, we are the first to propose an ILP-based optimal solution to the problem. We also propose a heuristic that performs within $2\times$ of the optimal solution on average while executing orders of magnitude faster.

Flexibility in SFC can be leveraged in other scenarios as well. For instance, in the event of substrate node or link failure, flexibility can be leveraged for restoring failed SFCs while minimizing backup footprint. Another interesting direction is to investigate how flexibility in SFC can be applied for re-optimizing resource allocation to alleviate bottlenecks.

REFERENCES

[1] "Network Functions Virtualisation - Introductory White Paper," Oct 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf
[2] F. Bari *et al.*, "Orchestrating virtualized network functions," *IEEE Trans. on Net. and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.
[3] J. G. Herrera *et al.*, "Resource allocation in nfv: A comprehensive survey," *IEEE Trans. on Net. and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.
[4] S. Mehraghdam *et al.*, "Specifying and placing chains of virtual network functions," in *Proc. of IEEE CloudNet*, 2014, pp. 7–13.
[5] ——, "Placement of services with flexible structures specified by a yang data model," in *Proc. of IEEE NetSoft*, 2016.
[6] S. Dräxler *et al.*, "Specification, composition, and placement of network services with flexible structures," *International Journal of Network Management*, vol. 27, no. 2, 2017.
[7] Y. Zhang *et al.*, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proc. of ACM SOSR*, 2017, pp. 143–149.
[8] C. Sun *et al.*, "Nfp: Enabling network function parallelism in nfv," in *Proc. of ACM SIGCOMM*, 2017, pp. 43–56.
[9] S. Ayoubi *et al.*, "A cut-and-solve based approach for the vnf assignment problem," *IEEE Trans. on Cloud Computing*, 2017.
[10] H. Moens *et al.*, "Vnf-p: A model for efficient placement of virtualized network functions," in *Proc. of CNSM*, 2014, pp. 418–423.
[11] B. Addis *et al.*, "Virtual network functions placement and routing optimization," in *IEEE CloudNet*, 2015, pp. 171–177.
[12] M. C. Luizelli *et al.*, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. of IFIP/IEEE IM*, 2015, pp. 98–106.
[13] R. Cohen *et al.*, "Near optimal placement of virtual network functions," in *Proc. of IEEE INFOCOM*, 2015, pp. 1346–1354.
[14] W. Ma *et al.*, "Traffic aware placement of interdependent nfv middleboxes," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.
[15] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proc. of ACM HotSDN*, 2013, pp. 127–132.
[16] D. Joseph *et al.*, "Modeling middleboxes," *IEEE network*, vol. 22, no. 5, 2008.
[17] A. Panda *et al.*, "Verifying reachability in networks with mutable datapaths." in *Proc. of USENIX NSDI*, 2017, pp. 699–718.
[18] S. K. Fayaz *et al.*, "Buzz: Testing context-dependent policies in stateful networks." in *Proc. of USENIX NSDI*, 2016, pp. 275–289.
[19] P. Kazemian *et al.*, "Header space analysis: Static checking for networks." in *Proc. of USENIX NSDI*, 2012, pp. 113–126.
[20] W. Wu *et al.*, "Automatic synthesis of nf models by program analysis," in *Proc. of ACM HotSDN*, 2016, pp. 29–35.
[21] ——, "Network function modeling and its applications," *IEEE Internet Computing*, vol. 21, no. 4, pp. 82–86, 2017.
[22] J. Sherry *et al.*, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Comm. Rev.*, vol. 42, no. 4, pp. 13–24, 2012.
[23] S. W. Brim *et al.*, "Middleboxes: Taxonomy and issues," 2002.
[24] N. Freed, "Behavior of and requirements for internet firewalls," 2000.
[25] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.
[26] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Comm. Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
[27] S. Kumar *et al.*, "Service function chaining use cases in data centers," *IETF SFC WG*, 2015.
[28] T. Benson *et al.*, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM IMC*, 2010, pp. 267–280.
[29] N. Spring *et al.*, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Comm. Rev.*, vol. 32, no. 4, pp. 133–145, 2002.
[30] L. Saino *et al.*, "A toolchain for simplifying network simulation setup," in *Proc. of SIMUTools*, 2013, pp. 82–91.
[31] A. Nucci *et al.*, "The problem of synthetically generating ip traffic matrices: initial recommendations," *ACM SIGCOMM Computer Comm. Rev.*, vol. 35, no. 3, pp. 19–32, 2005.
[32] "Wanos wan optimizer admin guide - hardware requirements." [Online]. Available: http://wanos.co/docs/docs/wanos-admin-guide/getting-started/hardware-requirements/

Factors Affecting Performance of Web Flows in Cellular Networks

Ermias Andargie Walelgne*, Setälä Kim*‡, Vaibhav Bajpai†, Stefan Neumeier†, Jukka Manner* and Jörg Ott†
*Aalto University, Finland, ‡ Elisa Oy, Finland
(ermias.walelgne | kim.setala | jukka.manner)@aalto.fi
†Technische Universität München, Germany
(bajpaiv | neumeier | ott)@in.tum.de

***Abstract*—Studies show that more than 95% of the traffic generated by smartphones typically consists of short-lived TCP flows towards websites. The content of such websites often is distributed across multiple servers which requires clients to resolve multiple DNS names and establish multiple TCP connections to fetch the webpage in its entirety. Studies have shown that network latency in a mobile network (attributed to DNS lookup and TCP connect times) contributes heavily to poor experience when browsing such websites. However, there is little understanding of the factors that contribute to high DNS lookup and TCP connect times. In this paper, we take this further by measuring the Domain Name System (DNS) lookup time and the TCP connect time to popular websites from ~25K subscribers of a cellular network operator in Finland. Using a month-long dataset (Oct 2016) of these measurements, we show that LTE offers considerably faster DNS lookup time compared to legacy cellular networks (such as HSPA+ and UMTS). We also show that the model of the device and the proximity of the DNS server to the subscribers impacts the DNS lookup time. Furthermore, the TCP connect time is also affected by the radio technology. We observe that LTE offers a significantly low latency profile such that the TCP connect time to popular websites is reduced by ~80% compared to legacy cellular networks. The presence of ISP caches also considerably improves TCP connect times. Using a `ping` test, we also observe that legacy radio technologies (such as HSPA+ and UMTS) suffer from higher packet loss than LTE.**

I. Introduction

The trend of users using mobile handheld devices to access the Internet shows a steady increase over the last years. If a user is on the move, these devices commonly use the cellular network to access the Internet. Huang *et al.* [1] show that the majority of network traffic (more than 95%), generated by smartphones typically consists of short-lived TCP flows towards websites. The content of such websites is often distributed across multiple servers, which requires mobile users to resolve multiple DNS names and establish multiple TCP connections to fetch the webpage in its entirety. Internet Service Provider (ISP)s such as T-Mobile [2] have shown that mobile users experience poor web-browsing usually due to high DNS lookup and TCP connect times. Similarly content providers such as Google report [3] that high network latency in a mobile network is contributed by multiple factors such as high DNS lookup, TCP connection and HTTP request times. These latency overheads usually incur before any actual data exchange happens. However, there have been few studies [4], [5] that quantify the factors that are responsible for higher DNS lookup and TCP connect times. This is largely because of lack of datasets with rich metadata information (such as the accessed radio technology during the measurement, the device model, *et al.*) that can help to identify those factors for mobile users in a cellular network. Using a month-long (Oct 2016) dataset (see § III) collected by an ISP from subscribers of a cellular network, we profile the performance of ~25K clients distributed across Finland (see Fig. 1) to understand the factors affecting performance in cellular networks. We focus on the performance of short web flows (such as DNS lookup and TCP connect times towards popular websites) that are driven more by latency than by network throughput. We also analyze the packet loss and RTT using more than 2M `ping` measurements towards `www.google.fi`. The performance over the home wireless network is not considered in this work. Towards this end, we provide three main contributions –

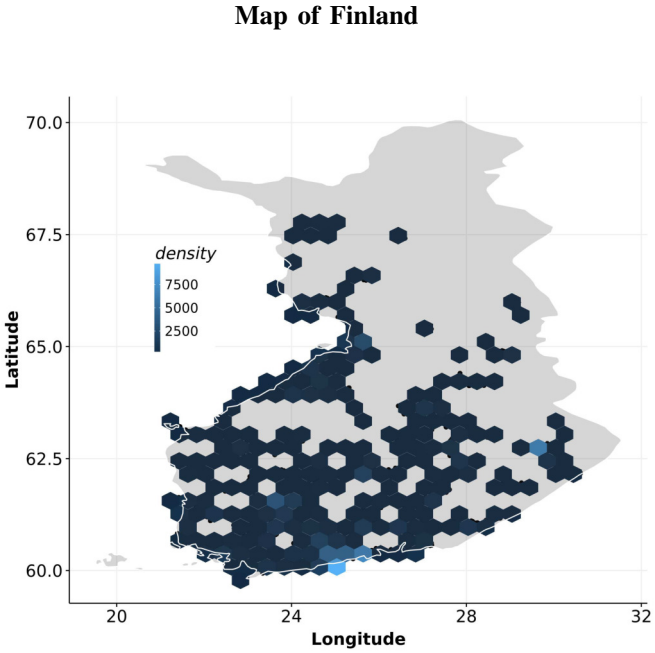


Fig. 1: **The geographical distribution of ~25K subscribers in Finland that participated in this measurement activity.**

– We observe ~2% DNS failures due to `BADVERS` or

ISBN 978-3-903176-08-9

BADSIG and YXRRSet errors. We show that packet loss can be underestimated in situations where a ping test sends less than 5 ICMP packets (§ IV) per instance. We also show that the legacy (such as HSPA+ and UMTS) technologies suffer from higher packet loss than Long-Term Evolution (LTE).

– We observe that TCP connect times are affected by radio technology used by the subscriber. LTE offers a significantly low latency profile (§ V) such that TCP connect times to popular websites are reduced by ~80% on LTE compared to legacy networks. Furthermore, we observe that LTE based data subscription plans do not have an impact on TCP connect and DNS lookup times.
– The device model (§ VI) and the DNS server's proximity to the subscriber has an impact on DNS lookup time. TCP connect times towards popular websites using the same radio technology are comparable, although their DNS lookup times (§ VII) exhibit a difference. ISP caches improve TCP connect times towards www.google.fi and www.youtube.com, while we do not observe cache deployments for www.facebook.com within our dataset. As a result only half of the TCP connections towards www.facebook.com completed within 50 ms, while up to ~80% of TCP connections towards www.google.fi and www.youtube.com completed within 50 ms.

The rest of the paper is structured as follows. We present related work in § II. The methodology for measuring DNS lookup time, TCP connection establishment time, packet loss and the resulting dataset are presented in § III. We then discuss the results in § IV to § VII. We describe the limitations in § VIII and conclude in § IX.

II. Related Work

We discuss previous work that investigates different factors affecting the performance within cellular networks. For instance, Rodriguez *et al.* [6] identified that multiple DNS lookup operations are one of the major cause for poor network throughput observed in an UMTS network. Xu *et al.* [4] measured DNS lookup time from user's mobile device to evaluate the performance of DNS resolvers and showed that placing the content close to a Gateway GPRS Support Node (GGSN) helps to speed-up content delivery. Jiang *et al.* [7] examined how large buffers in cellular networks contribute to significant TCP queuing delay. Huang *et al.* [1] used data collected from within a carrier's core network to study protocol level interaction of TCP over LTE. They show that TCP underutilizes (by more than 50%) the available bandwidth over LTE. A recent work by Nikravesh *et al.* [8], which is more related to our work, used two mobile apps to measure DNS lookup time to analyze performance variability across carriers and location. Nguyen *et al.* [9] showed how the performance of TCP over an LTE network is affected by handover and sudden load increase on the base station. They use a simulation environment and show that performance of TCP can degrade as the load on the carrier's cell increases.

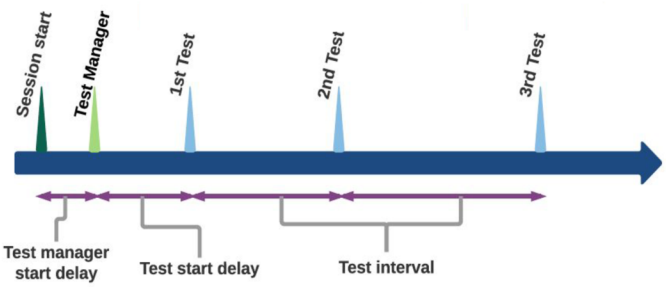


Fig. 2: **Measurement setup session. Immediately after the session starts, the Test Manager module is started. To minimize the possible overload on network or CPU (as many other application might run at the start of the network connection), the Test Manager Start Delay is set to 5 seconds. Test Start Delay is set to few seconds, where at least one measurement is conducted per session. The Test Interval is the interval between subsequent active tests (from ending time of the test until the next test starts). This interval relatively keeps increasing to avoid an excessive data collection.**

Our study covers a broader user-base with ~25K subscribers with ~20M measurements using both, the 3G and LTE network. It combines DNS lookup time, TCP connection establishment time and packet loss metrics that are collected from mobile users' vantage point. In this study, we analyze factors affecting the performance of short web flows in a mobile network which can then be used to identify the potential network performance bottlenecks. Our measurement is conducted towards www.facebook.com, www.google.fi and www.youtube.com. Marquez *et al.* [10] have recently shown that these websites share more than ~30% of traffic in the Orange's mobile network today. We also measure towards www.elisa.net (website of the ISP covered in this study), which covers a substantial fraction of mobile network traffic.

III. Methodology

The dataset we used for the analysis is collected by a carrier's network measurement platform based in Finland (see § VIII for limitations). It is a mobile application for iOS and Android installed on subscribers mobile devices [11] that conducts measurements tests on DNS lookup, TCP connection establishment and ping.

A. Measurement Setup

These measurement tests are executed inside a session as shown in Fig. 2. A session starts when a network interface that provides access to the Internet, is available and there is no interface claiming to yield a default route. The session ends in situations when the interface that was providing the default route to the Internet becomes unavailable or the client loses the assigned IP endpoint. Sessions are not periodic, but they are repeated when network conditions change. Such as,

when multiple interfaces claim to provide a default route to the Internet, and the 'best interface' [1] changes, the current session is terminated and a new session starts.

B. Measurement Tests

1) DNS Lookup Time: This test measures the time it takes to look up a Fully Qualified Domain Name (FQDN) from a DNS server and resolve it into an IPv4 address (see § VIII for limitations). The test allows one to specify a set of DNS servers and target DNS names. The DNS servers can either be statically configured or automatically assigned by the DHCP server. In our work, we measure the DNS lookup time of four popular websites: `www.google.fi`, `www.youtube.com`, `www.facebook.com` and `www.elisa.net`, as they are commonly known (see § VIII for limitations). Bajpai *et al.* [12] have shown that `www.google.*` websites are served by the same CDN and therefore exhibit similar latency behavior. As such, we use `www.google.fi` for our measurement study.

The test records the resolved DNS name and the IPv4 address of the DNS server. The IPv4 address of the client (majority of which are NATed and consequently receive an IP endpoint from the private [13] address space), the DNS lookup time (in milliseconds), device model type, the radio technology used during the test and the DNS response code indicating the success (or failure) of the test. A timeout of 30 seconds is used in situations where the DNS server is not reachable or the packet is lost. In such a situation, the client does not retry for a failed or timed-out request.

2) TCP Connect Time: This test measures the time it takes to establish a TCP connection (over IPv4) to a target website (over port 80) from the client device (see § VIII for limitations). The test starts when the client sends a `SYN` packet to a destination identified by a FQDN. It then subtracts this time value from the time of receiving a `SYN+ACK` packet from the server. This time difference does not include the DNS resolution time.

The test records the starting time of the test, FQDN of the destination host, destination port number, resolved IPv4 address of the destination host, TCP connect time, clients' device model type, the radio technology used during the test, and the success (or failure) of the TCP connection establishment.

3) RTT and Packet Loss: This test uses `ping` to measure the RTT and packet loss towards `www.google.fi` (see § VIII for limitations) using ICMP echo request packets. Each `ping` test sends an average of five to nine ICMP echo requests from clients to the target. The payload for each ICMP echo request is configured to be 16 bytes in size.

The test records the DNS name and the resolved IPv4 address of the target, the IPv4 address of the client (majority of which are NATed and consequently receive an IP endpoint from the private [13] address space), total elapsed time of the test, the number of `ping` tests, payload size of the ICMP echo request packet, the minimum, maximum and average RTT, the number of packets sent and received in the test, the response code indicating the success (or failure) of the execution, device model and the radio technology type used during the test.

[1]The interface with the lowest value of the metric attribute

TABLE I: **DNS, TCP and `ping` measurements by website.**

Website	DNS (#)	TCP (#)	`ping` (#)
`www.facebook.com`	3,471,440	4,572,298	-
`www.google.fi`	6,981,348	4,855,516	2,180,700
`www.youtube.com`	1,628,991	4,075,477	-
`www.elisa.net`	1,821,334	5,335,350	-

C. Dataset

The measurements are collected from ~25K subscribers of a cellular network provider based in Finland, geographically distributed as shown in Fig. 1. The dataset consists of ~14M samples of DNS lookup time, ~19M samples of TCP connect time and ~2M samples of `ping` measurements collected in October 2016. Table I provides details of the samples collected towards each target website.

IV. Failures

A. DNS Lookup

DNS based redirection techniques are used by content providers (such as Akamai [14]) to determine the location of the end-host and to redirect the contents to the closest content replica [4]. DNS errors may happen for various reasons including poor configuration errors [15], heavy load on the DNS server, and poor network link quality between server and clients. Such errors, if not managed well, could cause drastic damages as it happened in [16], where missing a terminating `'.'` to the DNS records of `.se` zone shutdowns a whole bunch of websites and news outlets in Sweden.

We use the DNS response code to determine the number of successful DNS responses and failures. About 86% of the DNS failures (which is about 2% from the total DNS lookup test) are `BADSIG` or `BADVERS` [17] (Bad OPT Version or TSIG Signature fails), indicating that a responder does not implement the version level of the request [18]. The second most frequent DNS failure code observed is `YXRRSet` [19] which means that the RR Set exists when it should not. Some other DNS failures such as `BADTIME` [20] (out of time windows) and `BADMODE` [21] (Bad TKEY Mode) also rarely happen. One reason for DNS failures to happen is a poorly configured DNS resolver. We noticed that out of all DNS failures that are observed, about 67% of the DNS lookup queries were sent towards the AS790 (Elisa) DNS resolver.

We observe that DNS lookup over LTE experiences about 1.9% of DNS failures, while over UMTS, HSPA and HSPA+ experience 3.4%, 3.9% and 2.7% DNS failures, respectively. Table II shows DNS failures by website. As it can be seen, these failure are almost evenly distributed over the different websites. There is a 2% DNS lookup failure variation between the `www.youtube.com` and `www.google.fi`

TABLE II: DNS Failures per website using the LTE network.

Website	Failures (%)
www.facebook.com	2.16
www.google.fi	0.96
www.youtube.com	2.99
www.elisa.net	2.74

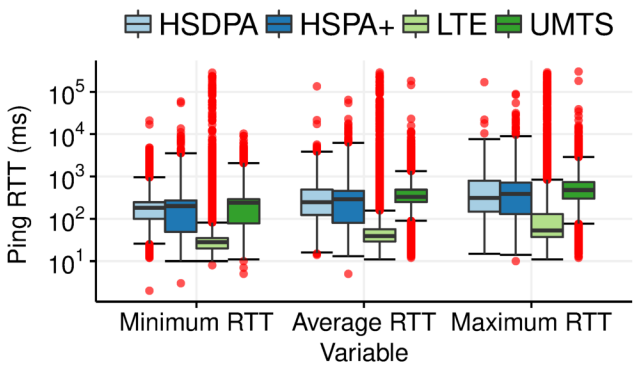


Fig. 3: **Minimum, average and maximum RTT values split by radio technology for a `ping` test towards `www.google.fi`.**

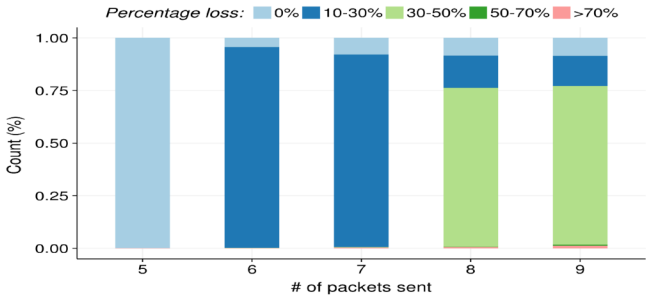


Fig. 4: **Percentage packets loss across the number of packets sent.**

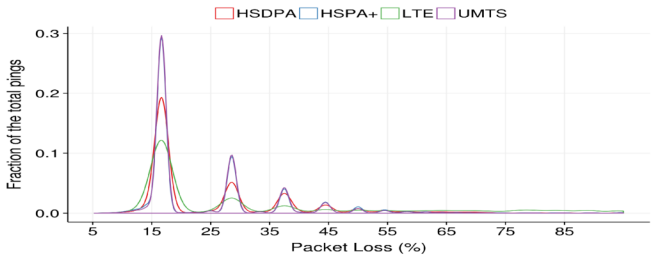


Fig. 5: **Distribution of packet loss as the fraction total `ping` by radio technology type.**

domain names. A variation is also observed on the DNS lookup time (see section V).

B. TCP Connect Time

From the over all TCP connect time measurement dataset a quite small fraction of the dataset (~1.07%) shows a TCP connect time failure. Out of theses errors the majority of them (~0.98%) were raised due to "connection refused" error type. As per RFC-793 [22] this error can happen if the client received a `RST` bit form the server, indicating that the TCP connection must be reset. This could be either due to an intervening firewall that blocks the `SYN` packet or if the destination server is refusing the TCP connection. We observe that out of all measurements conducted using LTE, HSPA+, HSPA and UMTS 1.0%, 1.2%, 2% and 1.5% experience failures, respectively. We observe a similar failure distribution using the same radio technology for each websites.

C. Packet Loss Using `Ping` Test Measurement

Fig. 3 shows that minimum, average and maximum RTT using LTE network are 27, 38, 51 ms, respectively. We observe that 90% of the average `ping` test measurements towards `www.google.fi` using LTE have a RTT time less than 100 ms. While other legacy 3G technologies are quite slow with than 200 ms RTT.

We observer that ~14.98% of tests in ping measurement (see Table I) have at least one packet loss. Packet loss is calculated from the number of Echo Requests (number of packets sent) and Echo Responses (number of packets received). A packet loss happens if the number of received packets is less than the number of packets sent. We observe that most of the packet loss happens, if the number of packets sent at every `ping` test instance is more than five Echo Requests. On the one hand, of about 80% of the `ping` test instances that are measured by sending five individual Echo requests, only 0.33% of the `ping` test instances suffer from packet loss. On the other hand, `ping` test instances that were measured by sending more than five Echo Requests suffer from the highest number of packet loss. For instance, measurements that have 6 or 7 Echo Requests suffer with a higher number of packet loss, covering 57% and 19% of the the total `ping` packet loss, respectively. Fig. 4 shows the percentage of packets loss across the number of packets sent at each instance of the `ping` test.

We noticed that more than 77% of the `ping` test measurements were conducted using the LTE network. Only 2.4% of them lost at least a single packet. The second and the third highest number of `ping` measurements are carried over HSPA+ and UMTS, each sharing 11.18% and 8.28% from total measurement, respectively. More than 50% of the `ping` tests on HSPA+ experience at least one packet loss. The `ping` test using the UMTS network suffers from the highest packet loss, reaching up to 65%.

Fig. 5 shows the distribution of `ping` test fraction that experiences various percentage of packet loss when tested with a different radio technology. For example, with LTE, more than 0.1 fractions of `ping` tests suffer from 15% packet loss. Note that, both Fig. 5 and Fig. 4 show the `ping` test by excluding the cases in which the packet loss was zero percent.

Going forward (sections V - VII-A) we focus on different

TABLE III: DNS & TCP Measurements by radio technology.

Radio Technology	DNS (%)	TCP (%)
LTE	68.94	69.59
HSPA+	10.59	10.23
HSPA	2.41	2.41
UMTS	14.51	14.72
Others	3.55	3.05

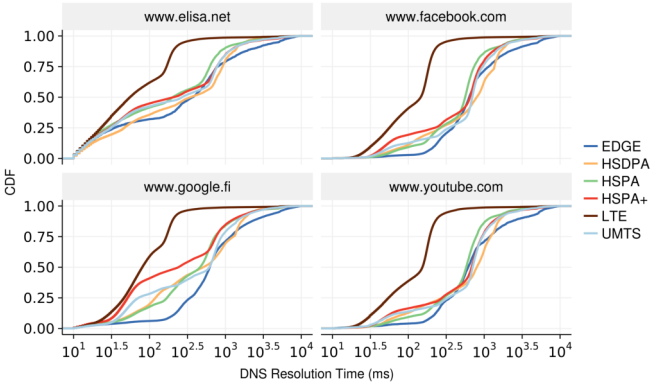


Fig. 6: **DNS response times by radio technology: LTE exhibits significantly lower latency.**

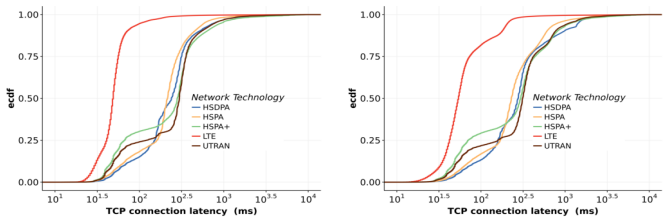
factors impacting DNS lookup and TCP connect times.

V. Radio Technology

In today's cellular network environment, there is quite a range of radio technologies with different levels of performance. These radio technologies including LTE, HSPA+, HSPA and UMTS have a various range of bandwidth performance. Most of the today's mobile devices are equipped with all of these radio technologies. We analyze how DNS lookup and TCP connect time varies across different radio technologies.

DNS Lookup Time: Fig. 6 shows that there is a clear DNS lookup time difference between the radio technologies. There is a fast DNS resolution time in DNS lookup for recent network technologies such as LTE and HSPA+ and a considerably long resolution time for anterior technologies such as HSPA and UMTS.

The LTE network technology consistently shows the best DNS resolution performance on all of the four tested websites. The median difference between LTE and UMTS for resolving `www.google.fi` is 370 ms. Fig. 6 also shows the poor performance of earlier radio technologies such as EDGE, that takes more than half of a second (624 ms) to resolve the IP address of `www.google.fi`. The percentaged difference of DNS response time between LTE and other radio technologies varies among the different domain names. Fig. 6 shows that 50% of the requests send to `www.elisa.net` are resolved in less than 500 ms, irrespective of the radio technology type. Except using the LTE network, only 25% to 30% of the DNS queries send to `www.facebook.com` and `www.youtube.com` are resolved in less than 500 ms. In other words more than 70% of DNS lookup queries send to `www.facebook.com` and `www.youtube.com` took more than 500 ms to get back the resolved IP address.



(a) `www.youtube.com` (b) `www.google.fi`

Fig. 7: **TCP connect time towards `www.youtube.com` and `www.google.fi` by radio technology. The distribution exhibits similar pattern for `www.elisa.net` and `www.facebook.com`, too.**

The probability of resolving `www.google.fi` below 100 ms using LTE and HSPA+ radio technology is 65% and 58%, respectively, which is a difference of 7%. Whereas, the probability of resolving `www.youtube.com` below 100 ms using LTE and HSPA+ radio technology is 36% and 23%, respectively, which is a difference of 13%. For most of the 3G and 4G technologies, about 50% of the time, the DNS resolution of `www.google.fi` takes more than 500 ms.

TCP Connect Time: We study the performance variation among radio technologies by comparing the latency to reach a give website address through TCP. Fig. 7 shows TCP connect times of `www.youtube.com` and `www.google.fi` using different radio technologies. Similar to the DNS lookup latency, LTE outperforms all other radio technologies. For example, about 92% of the TCP connect time tests using LTE have less than 100 ms latency. Whereas only about 28% of the 3G based TCP connect time tests are below 100 ms. The median TCP connect time of `www.youtube.com` under LTE and Legacy technology is 50 ms and 251 ms, respectively. Thus, LTE reduces the TCP connect latency by 80%. The measurement distribution of TCP connect time and DNS lookup test by radio technology is shown in Table III.

Given that we know the difference between LTE and legacy radio technologies, going forward, we only look at factors affecting performance on the LTE network.

A. LTE Subscription Plan

We use few randomly selected sample clients' data subscription plan as a reference to study the impact of data subscription plan for DNS lookup and TCP connect time latency. The clients' data plan is classified into 2 packages based on the downlink and uplink speed limits. These are `4G` and `4G-super` for the upper-downlink limit of 25 and 80 Mbits/s, respectively. Note, we only consider LTE in this section.

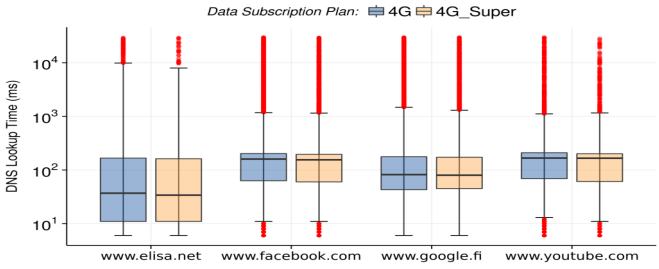


Fig. 8: **DNS response times by subscription plan.**

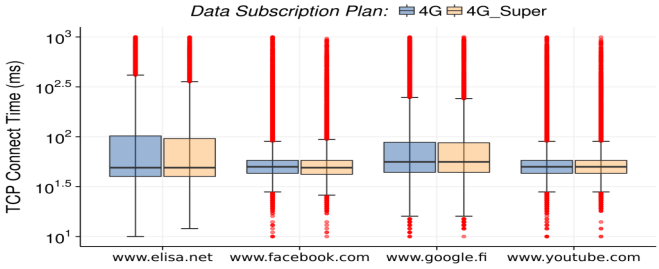
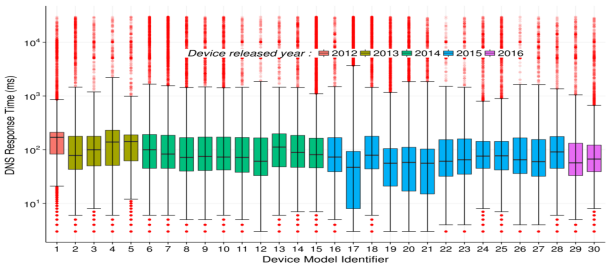
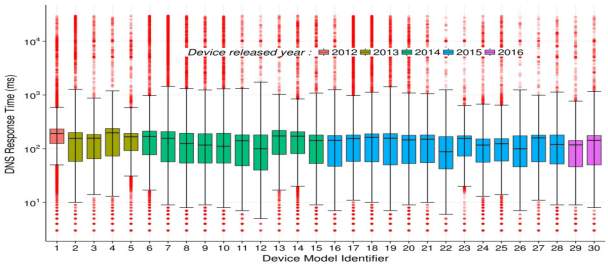


Fig. 9: **TCP connect time by subscription plan.**



(a) `www.google.fi`



(b) `www.facebook.com`

Fig. 10: **DNS response time of `www.google.fi` (above) and `www.facebook.com` (below) across device models as measured over LTE. Order by device models' release year.**

DNS Lookup time: Fig. 8 depicts the DNS response time per users' data subscription plan for each radio technology. The graph shows that the clients' data subscription plan does not actually contribute to the DNS lookup time performance.

TCP Connect Time: As depicted in Fig. 9, the data subscription type has a very small impact on the TCP connection establishment time.

VI. Device Models

We analyzed the impact of different device model types and year of release for both TCP connect time and DNS lookup time performance.

DNS Lookup Time: Fig. 10 shows the DNS response time of the top 30 device models, ordered by device models' release year. All the devices are capable of using both, 3G and LTE radio technologies. The selected devices were using the LTE network during the DNS test session. The model names are substituted with the index number to ensure anonymity. The y-axis reflects the DNS lookup time value of at least 10K individual tests for each device that subscribed to a single network operator. We can observe that there is a significant difference in DNS resolution time among device models. For instance, observing the median value of devices released in the year 2015, it appears that the device model *#17* has the highest DNS resolution time, whereas the device model *#22* has a relatively short DNS lookup time for resolving the domain name `www.facebook.com`. The standard deviation (not shown in the plot) of the DNS lookup time across the 30 devices is also highly variable, ranging from 622.45 to 3891.36 ms. The ANOVA [23] F-test for DNS response time is also significant (P-value of 0.0001), asserting that the DNS resolution time is indeed affected by device model type.

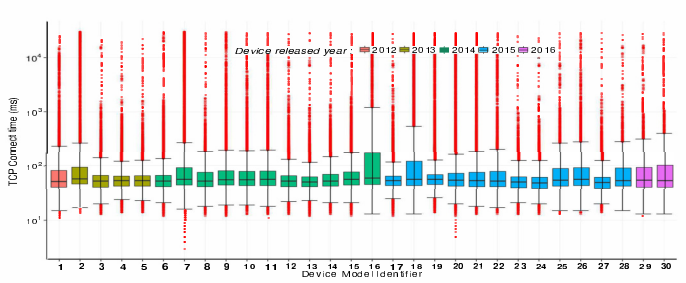
To further explore this, we conduct a manual inspection to some of the devices by minimizing the variance such as by fixing the subscribers location and time. From the manual inspection, we observe that few device models consistently show a poor resolution time performance in both LTE and 3G radio technologies. We also observe that devices which have larger internal memory and storage capacity are relatively faster conducting a DNS lookup.

TCP Connect Time: The impact of various device model types for TCP connect time latency is very small, especially when it is compared to the DNS lookup time. As shown in Fig. 11, except few devices the median latency among device types when tested towards `www.google.fi` and `www.facebook.com` is less than 100 ms. The device model's release year has also no direct impact on the TCP connection establishment time variation.

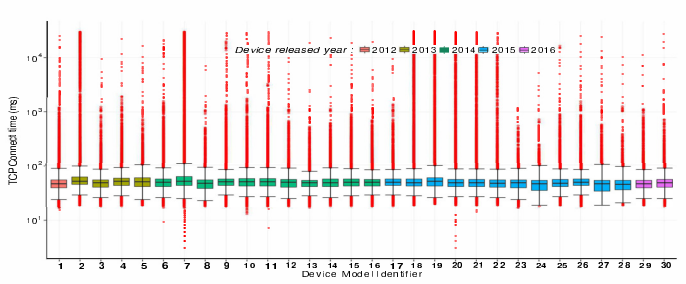
VII. Websites

DNS Lookup Time: Fig. 12 shows that the DNS lookup time significantly varies among different websites, using the same radio technology (LTE) that has been accessed during the DNS test. The DNS lookup times of `www.youtube.com` and `www.facebook.com` are significantly slower than the ones of `www.google.fi` and `www.elisa.net`. One cause is that the A entries for `www.google.fi` and `www.elisa.net` (ISP's website) are more likely to be cached by DNS resolvers than `www.youtube.com` and `www.facebook.com`.

TCP Connect Time: The TCP connection time is one important measure for websites download time and user satisfaction. Prior work [24] has shown that about 17% of the



(a) `www.google.fi`



(b) `www.facebook.com`

Fig. 11: **TCP connect time for `www.google.fi` (above) and `www.facebook.com` (below) across device models as measured over LTE.: Order by device models' release year.**

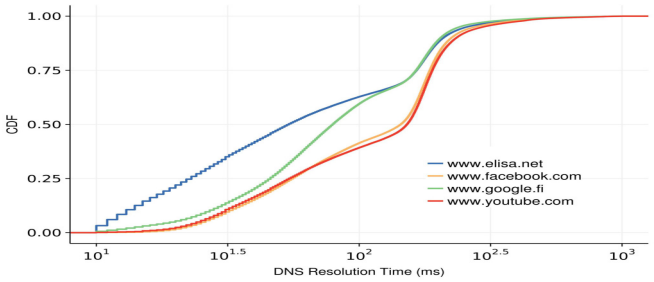


Fig. 12: **DNS response time towards websites using LTE; tested towards different DNS resolvers. Note that the variation almost stays the same if we fix it to a single DNS resolver.**

users are impatient to wait if the response time of a given website is greater than 5 seconds. Thus, we analyze the TCP connectivity time for different websites.

The time elapsed between sending the SYN packet to open the TCP socket and receiving the SYN+ACK response to selected website addresses is shown in Fig. 13. We observe that the majority of TCP connection latencies using LTE range from 20 to 200 ms, irrespective of the website's address. For instance, about 97% of the TCP connections to `www.facebook.com` are completed in less than 200 ms.

We can see that 90% of the time, `www.facebook.com` and `www.youtube.com` can be reached in less than 100 ms from a client's device. Whereas, for `www.google.fi` and `www.elisa.net`, only 80% and 76% of the TCP

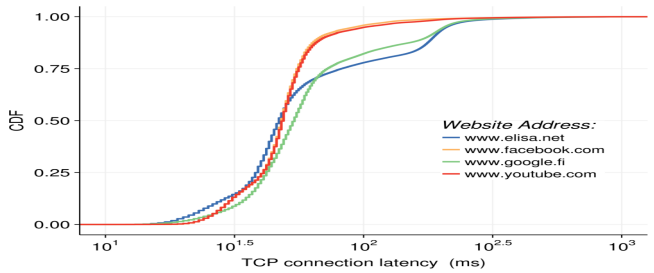


Fig. 13: **TCP connect time towards websites under LTE.**

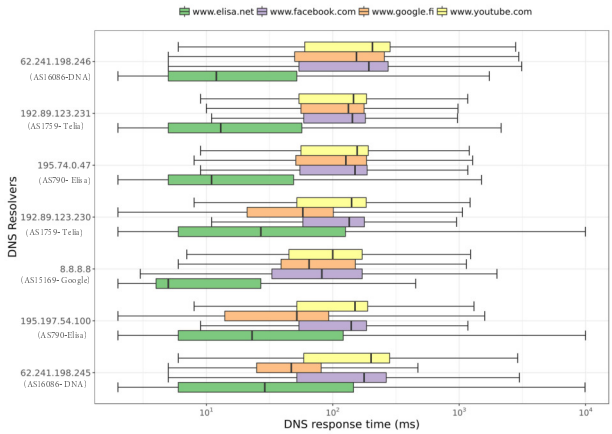


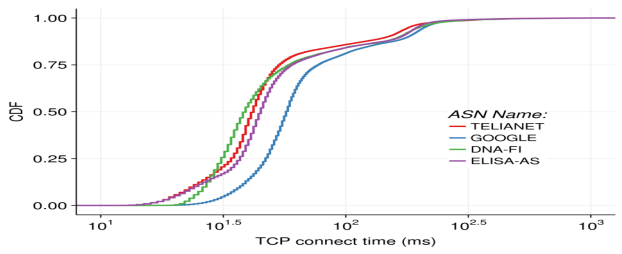
Fig. 14: **DNS response time by resolver IP address using LTE.**

connection test are below 100 ms, respectively.

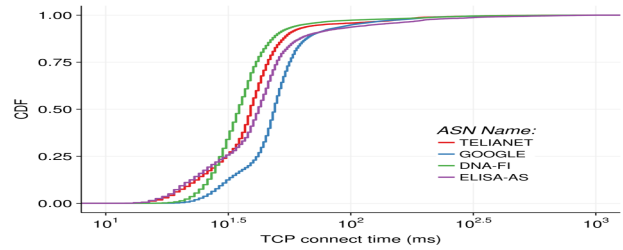
A. Destination Autonomous System Number (ASN)

DNS Lookup Time: Previous studies show that cellular DNS servers can yield faster DNS lookup time than public DNS resolvers [5]. In light of this, we compare the capability of different DNS servers to resolve a domain name to an IP address. Fig. 14 shows the DNS lookup time of different resolvers per website address. Each of these DNS resolvers IP has more than 10K measurements. We can see that some cellular network DNS servers have a faster DNS lookup time for `www.google.fi` than Google DNS servers. We also notice that there is a significant variation between DNS resolvers belonging to the same ISP. For instance, two DNS resolvers inside AS790, `"EUNET. FI"` of two different IP entries 195.74.0.47 & 195.197.54.100, have 133 ms and 51 ms (median) to resolve `wwww.google.fi` using the LTE network. This variation might happen due to the closer proximity of the DNS resolver to the ISP network [25].

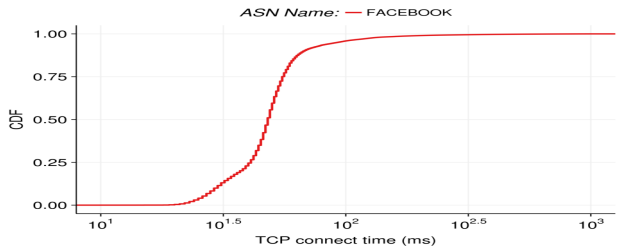
TCP Connect Time: For a better network traffic management and performance optimization, network operators may deploy proxy servers between the client and the target destination server [26]. We observe that a proxy or a cache server between the client and the true destination host server may acknowledge the TCP socket request first [27], [28]. This has the advantage of decreasing TCP connection time in the



(a) www.google.fi



(b) www.youtube.com



(c) www.facebook.com

Fig. 15: **TCP connect time towards `www.google.fi`, `www.youtube.com` and `www.facebook.com` by destination ASN from LTE networks.**

order of milliseconds (as shown in section VII-A). We use the RIPE [29] service to map the resolved IP address of the websites to an ASN value.

Fig. 15 (a) shows that the latency for a TCP connection to reach the website `www.google.fi` varies based on the ASN number for the same radio technology (LTE). We can see that subscribers served by the ISP network manage to reach the `www.google.fi` website faster than the request sent to Google-owned web-servers. One possible reason of the low TCP connection time for those hosted by ISP would be that web proxies are used to improve browsing performance response [30]. This means, if a TCP connection request is sent to the actual server, the proxy, which is installed between the client and the true destination server, may acknowledge the socket request before passing it to the destination server.

Fig. 15 (b) shows that TCP connect time latency towards `www.youtube.com` varies by the ASN value using the same radio technology (LTE). Clients served by the ISP network have managed to reach the `www.youtube.com` website in short time. This indicates that pushing the content close to the subscriber could potentially reduce the end-to-

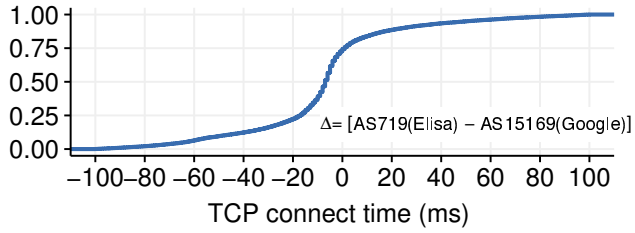


Fig. 16: **TCP connect time towards `www.google.fi` showing the latency difference between ISP cache - Elisa (AS719) and CDN - Google (AS15169) using LTE. Delta is the TCP connect time difference between Elisa and Google when the same user is getting a reply from the two network within a one hour time window.**

end latency by more than 20% compared to the requests sent to YouTube-owned web-servers; this is equal to [31], which points that caching improves the fetch time of small files. Fig. 15 (c) shows all TCP connection requests sent to `www.facebook.com` were served by a single ASN. Since `www.facebook.com` does not hit any caches in the ISP network, TCP connect time towards `www.facebook.com` is substantially slower than towards `www.youtube.com` and `www.google.fi`. This is shown in Fig. 15.

$$\Delta t(ct) = isp(ct) - cdn(ct) \quad (1)$$

We use Eq. 1 to calculate the TCP connect time difference between an ISP cache and CDN. For this, we created a pair of CDN and ISP per user within a one-hour time frame. First, we grouped the dataset by the user, ASN and one-hour window. If there is more than one measurement by a given user in a combination, we take the mean value. Then, we keep the ones that have pairs (in this case Google and Elisa). Fig. 16 shows the distribution of difference in TCP connect times between two destinations, where values on the negative scale indicate that ISP cache is faster. We observe that about 70% of TCP connect time towards `www.google.fi` achieve lower latency when they hit ISP cache.

VIII. Limitations

The measurements only consist of clients based in Finland using IPv4. The only measured services are those that run on port 80. The websites chosen are the most commonly used websites (except `www.elisa.net`) following the Alexa [32] website ranking. The `ping` measurements are conducted only towards `www.google.fi`. As such, it is not known whether and how the observations would differ from a different client base per country and towards different websites or a different services on the Internet. However, these three websites (`www.facebook.com`, `www.google.fi` and `www.youtube.com`) have a high probability of reflecting the majority of the mobile web-user-experience as they generate a considerable size of network traffic in mobile networks [10].

IX. Conclusion

We presented an analysis on factors that affect DNS lookup time and TCP connect time towards popular websites in cellular networks. We showed that DNS lookup time significantly varies for different websites, even when the same radio technology is accessed during the measurement. We showed that caches closer to the ISP could significantly improve TCP connect time. Also, the proximity of DNS server to the subscriber has a higher impact on DNS lookup time performance. We also observed that LTE offers considerably low latency compared to legacy radio technologies. We show that packet loss can be underestimated in situations where a `ping` test sends less than 5 ICMP packets per instance. Thus, we recommend that a packet loss analysis based on the `ping` test should consider increasing the number of packets per `ping` test instance for better results.

Acknowledgements

This work is partially funded by the FP7 Marie Curie Initial Training Network (ITN) METRICS project (grant agreement No. 607728).

References

[1] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance," ser. ACM SIGCOMM, 2013, pp. 363–374. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486006

[2] J. Hui and K. Lau, "T-Mobile QoE Lab: Making Mobile Browsing Faster and Open Research Problems," ser. ACM MobiCom, 2013, pp. 239–242. [Online]. Available: http://doi.acm.org/10.1145/2500423.2504585

[3] I. Grigorik. (2013) Performance of Wireless Networks. [Online]. Available: https://hpbn.co

[4] Q. Xu, J. Huang, Z. Wang, F. Qian, A. Gerber, and Z. M. Mao, "Cellular Data Network Infrastructure Characterization and Implication on Mobile Content Placement," ser. ACM SIGMETRICS, 2011, pp. 317–328. [Online]. Available: http://doi.acm.org/10.1145/1993744.1993777

[5] J. P. Rula and F. E. Bustamante, "Behind the Curtain: The Importance of Replica Selection in Next Generation Cellular Networks," ser. SIGCOMM, 2014, pp. 135–136. [Online]. Available: http://doi.acm.org/10.1145/2619239.2631465

[6] P. Rodriguez, S. Mukherjee, and S. Ramgarajan, "Session Level Techniques for Improving Web Browsing Performance on Wireless Links," ser. WWW, 2004, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/988672.988690

[7] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling Bufferbloat in 3G/4G Networks," ser. IMC, 2012, pp. 329–342. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398810

[8] A. Nikravesh, D. R. Choffnes, E. Katz-Bassett, Z. M. Mao, and M. Welsh, "Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis," ser. PAM, 2014, pp. 12–22. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-04918-2_2

[9] B. Nguyen, A. Banerjee, V. Gopalakrishnan, S. Kasera, S. Lee, A. Shaikh, and J. Van der Merwe, "Towards Understanding TCP Performance on LTE/EPC Mobile Networks," ser. AllThingsCellular, 2014. [Online]. Available: http://doi.acm.org/10.1145/2627585.2627594

[10] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, C. Ziemlicki, and Z. Smoreda, "Not All Apps Are Created Equal: Analysis of Spatiotemporal Heterogeneity in Nationwide Mobile Service Usage," 2017. [Online]. Available: URI:http://eprints.networks.imdea.org/id/eprint/1710

[11] (2017) Mobiili OmaElisa. [Online]. Available: https://verkkoasiointi.elisa.fi/

[12] V. Bajpai and J. Schönwälder, "IPv4 versus IPv6 - who connects faster?" ser. IFIP, 2015, pp. 1–9. [Online]. Available: https://doi.org/10.1109/IFIPNetworking.2015.7145323

[13] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918 (Best Current Practice), RFC Editor, Fremont, CA, USA, pp. 1–9, Feb. 1996, updated by RFC 6761. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1918.txt

[14] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1842733.1842736

[15] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, "Impact of Configuration Errors on DNS Robustness," ser. SIGCOMM, 2004. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015503

[16] P. McNamara. (2009, October) Missing dot drops Sweden off the Internet. [Online]. Available: http://www.networkworld.com/article/2251220/lan-wan/missing-dot-drops-sweden-off-the-internet.html

[17] P. Vixie, "Extension Mechanisms for DNS (EDNS0)," RFC 2671 (Proposed Standard), RFC Editor, 1999, obsoleted by RFC 6891. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2671.txt

[18] ——, "Extension Mechanisms for DNS (EDNS0)," pp. 1–7, 1999. [Online]. Available: https://doi.org/10.17487/RFC2671

[19] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)," RFC 2136 (Proposed Standard), pp. 1–26, 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2136.txt

[20] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)," RFC 2845 (Proposed Standard), RFC Editor, 2000, updated by RFCs 3645, 4635, 6895. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2845.txt

[21] D. Eastlake 3rd, "Secret Key Establishment for DNS (TKEY RR)," RFC 2930 (Proposed Standard), RFC Editor, 2000, updated by RFC 6895. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2930.txt

[22] J. Postel, "TRANSMISSION CONTROL PROTOCOL," Internet Requests for Comments, RFC Editor, RFC 793, September 1981. [Online]. Available: https://tools.ietf.org/html/rfc793

[23] J. J. Faraway, "Practical regression and ANOVA using R." 2002. [Online]. Available: https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf

[24] Gomez. (2010) When seconds count. [Online]. Available: http://ftp.software.ibm.com/software/au/downloads/GomezWebSpeedSurvey.pdf

[25] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS Resolvers in the Wild," ser. ACM IMC, 2010, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879144

[26] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the Edge Network," ser. ACM SIGCOMM, 2010, pp. 246–259. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879173

[27] M.-C. Roşu and D. Roşu, "An Evaluation of TCP Splice Benefits in Web Proxy Servers," ser. WWW, 2002, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/511446.511449

[28] A. Pathak, A. Wang, C. Huang, A. G. Greenberg, Y. C. Hu, R. Kern, J. Li, and K. W. Ross, "Measuring and Evaluating TCP Splitting for Cloud Services," ser. PAM, 2010, pp. 41–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12334-4_5

[29] RIPE. (2016) RIPE Stat API. [Online]. Available: https://stat.ripe.net/

[30] P. Rodriguez and V. Fridman, "Performance of PEPs in Cellular Wireless Networks," ser. Web Content Caching and Distribution. Springer, 2004, pp. 19–38. [Online]. Available: https://link.springer.com/chapter/10.1007/1-4020-2258-1_2

[31] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. R. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," ser. PAM, 2015, pp. 262–276. [Online]. Available: https://doi.org/10.1007/978-3-319-15509-8_20

[32] (2017) Alexa: The top 500 sites on the web. [Online]. Available: https://www.alexa.com/topsites

Measurement and Analysis of the Reviews in Airbnb

Qian Zhou[1,2], Yang Chen[1,2], Chuanhao Ma[1,2], Fei Li[1,2], Yu Xiao[3], Xin Wang[1,2], Xiaoming Fu[4]
[1]School of Computer Science, Fudan University, China
[2]Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China
[3]Department of Communications and Networking, Aalto University, Finland
[4]Institute of Computer Science, University of Goettingen, Germany
Email: {chenyang, xinw}@fudan.edu.cn, yu.xiao@aalto.fi, fu@cs.uni-goettingen.de

***Abstract*—Airbnb, a recently emerged online lodging service that allows house and apartment dwellers to lease out their premises to short-term renters like tourists, is reconstructing the value chain of the traditional hotel industry. It works as a platform that connects hosts and travelers and facilities their interaction and exchange. Studying this service could shed light on understanding the emerging sharing economy from a user-centric perspective. In this work, we collect the profiles of 43.8 million Airbnb users, and analyze the reviews they published online. We model the interactions between Airbnb users using a review graph, and study their mobility patterns by investigating their reviews. To the best of our knowledge, our work is the first measurement study of massive Airbnb users on a global scale, and it provides insights of their activities in both cyberspace and the physical world.**

I. Introduction

Firstly launched in 2008, Airbnb becomes a popular online service for listing and renting short-term lodging in residential properties around the world today. In January 2018, Airbnb has more than 3 million listings in 65,000 cities and 191 countries[1]. Besides online booking, Airbnb also facilitates the social interaction between tens of millions of users. For example, users can communicate using a private messaging service, and can share their lodging experience through posting reviews for other users publicly. Understanding the user behavior is essential for improving user experience. However, so far there lacks a comprehensive study of Airbnb user behavior.

Each Airbnb user has a personal profile, including demographic information like home country as well as user reviews. In this paper, we adopt a data-driven approach to analyze Airbnb user behavior. We crawled the profile pages of almost all – if not all – Airbnb users (as of Nov. 8, 2015), and collected their demographic information and all the published reviews. Based on the massive amount of data we have collected, we analyze the Airbnb user behavior from the following perspectives.

First, we conduct a demographic analysis of Airbnb users based on several key fields of user profiles, including home country, verification status and their roles in apartment leasing. We find that Airbnb is getting globally recognized, although most users are still from North America and Europe.

Second, we focus on the visible interactions between hosts and guests, which are revealed by public reviews. We model the interactions with a global *review graph* G, and describe them with a number of classic graph metrics. By examining the evolution of the review graph from 2008 to 2015, we discover that more and more users have been added to a giant weakly connected component which covers at least 98% users in G.

Last but not least, we dive into the mobility patterns of Airbnb users. After studying the users' movements from both spatial and temporal aspects, we figure out the time and location preferences in users' traveling. Also, based on our results of sentiment analysis, a majority of users are satisfied with their lodging experiences.

II. Data Collection and Preprocessing

A. Data Collection

In our study, we aim to obtain a complete view of Airbnb user behavior. Therefore, instead of using such a subset of users for study, we have crawled all 43.8 million Airbnb users' personal profiles including all the published reviews. Due to the strict per IP address rate limit, it becomes challenging to crawl all the user data in a short time. We address this issue as follows. Firstly, each Airbnb user has a unique numeric UID. The UID is assigned sequentially, i.e., a user registered earlier will get a smaller UID. For each user, we can access her profile page via the URL https://www.airbnb.com/users/show/UID. When we registered a new account on Sep. 25, 2015, we got the up-to-date maximum UID, i.e., 45063045. Secondly, we divided the ID range [1, 45063045] evenly into 185 chunks, and launched 185 virtual instances on the Microsoft Azure platform to crawl the personal pages simultaneously. Each of these instances has an unique IP address. The crawling process was run from Sep. 25, 2015 to Nov. 8, 2015. Except few unused IDs, we have obtained 43.8 million users' profiles and all the published reviews. Note that we respect the privacy of Airbnb users. Only publicly accessible data are crawled.

B. Data Preprocessing

We derive the interactions between Airbnb users from the published reviews, and model the interactions with a social

[1]https://www.airbnb.com/about/about-us?locale=en

ISBN 978-3-903176-08-9

graph. We call the social graph "review graph". The reviews can be classified into two categories, including the reviews from guests and the ones from hosts. According to [8], for more than 70% of online bookings through Airbnb, the users have published reviews for the visits. Therefore, it is feasible to profile the Airbnb user behavior such as mobility patterns and social interactions based on the analysis of user reviews.

We denote the review graph by $G = (V, E)$. Each node in the node set V represents an Airbnb user. Two nodes are connected with a directed edge, if one of the users has hosted the other one and at least one of them has posted reviews. For example, if user A has stayed in user B's apartment, A might post a review on B's profile page from the guest's perspective. Meanwhile, B might post a review from a host's perspective. If either A or B has posted a review online, there will be a directed edge (v_A, v_B). All the edges form the edge set E. When building the review graph, we exclude the users who have never posted or received any review. The resulting review graph includes 19,341,495 nodes and 17,553,551 edges.

As we are interested in the yearly temporal evolution of the review graph, we need to know when each node and edge was created. Because the registration time (year and month) of each Airbnb user is published on the user's profile page, the creation time of each node can be obtained directly from there. The creation time of an edge depends on when the reviews are published. If a user has visited another one for several times and has posted reviews for more than one visit, we set the creation time of the edge as the year when the first review was published. We derive the year information from reviews following three steps. (1) We obtain the year information directly from the reviews when possible. There are two types of reviews, one from the guest and the other from the host. On each Airbnb user's profile page we can find the published time information (year and month) of the latest 7 reviews of each type. The reviews published earlier are listed in a reverse chronological order, but their published time information are hidden. Among all the users who have posted or received at least one review, only 3.32% of them have received more than 7 reviews from guests, and 1.74% of them have received more than 7 reviews from hosts. Still, 30.67% of reviews do not have the time information. For these reviews, we infer their published time in the following two steps. (2) If the host and the guest have made "mutual reviews", which means they have written reviews for each other, we can assume a short time interval between the reviews since Airbnb only allows a user to write a review for a trip within 14 days after checkout. To validate this assumption, we examine all the mutual reviews with timestamps. The results show that 97.8% of them were published in the same month, while 99.3% of them were published in the same year. Given a pair of mutual reviews, if one of them has a timestamp, it is very likely that the other one was published in the same year. With this feature, we are able to estimate the published year of 22.77% of all the reviews. (3) As all the reviews are listed in a reverse chronological order, we utilize this feature to estimate the range of the published year of reviews. For example, three reviews were published in order. If both the earliest and the latest ones were estimated to be published in year 2009, the middle one must be published in 2009 as well. With this feature, we manage to estimate the exact published year of 6.03% of edges. For the last 1.87% edges, we assign each of them a randomly generated year within the estimated time range.

Besides the author and published year of reviews, we also look into the content of each review. We conduct sentiment analysis of all the reviews written in English, which covers 92.66% of all the published reviews. We use a natural language processing (NLP) library called NLTK [1] to extract users' sentiment information from reviews. Based on the output of NLTK, we follow the VADER algorithm to calculate a sentiment score for each review [12]. VADER is designed for sentiment analysis of social media content. The sentiment score for each review ranges from -1 to 1. A score of 1 means the review is strongly positive, -1 means the review is very negative, and 0 indicates the review is neutral. Among all the reviews written in English, 97.13% of them are positive, 1.98% of them are neutral, and only 0.89% of them are negative. In other words, nearly all the reviews written in English are positive about the lodging experience.

III. Data Analysis

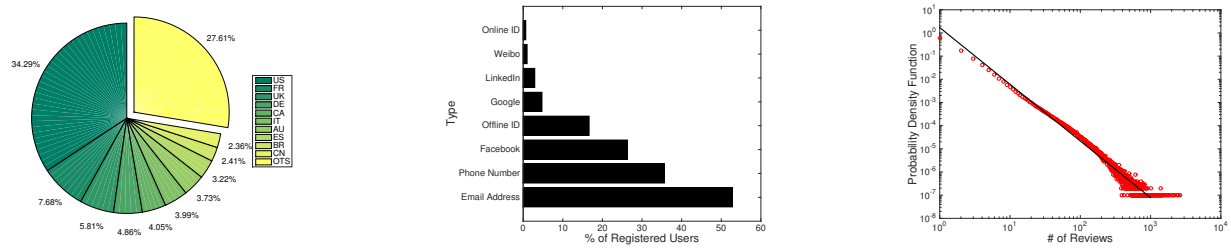
This work aims at providing insights on the Airbnb user behavior based on the analysis of personal profiles including published reviews. We analyze the crawled user data from the following three aspects. Firstly, we performance a comprehensive demographic analysis in § III-A to reveal the composition of Airbnb users. Secondly, we model the social interactions between users with a review graph, and analyze the static and dynamic characteristics of the review graph in § III-B. Thirdly, we investigate the mobility patterns of Airbnb users in § III-C.

A. Demographic Analysis

1) Statistics: The personal profile of a typical Airbnb user includes several information fields, such as location, verified ID, and "About Me". In addition, a user can request to become a verified user, in order to get a "V" badge displayed on her profile page. A small number of hosts satisfying certain requirements can also receive the "superhost" badge, which will also be shown on the user's profile page.

User Location Referring to the "location" indicated on the personal profile, we identify the home country of 86.20% of registered Airbnb users. As shown in Fig. 1(a), 34.29% of users come from the United States. In total, nearly 60% of users come from one of the 5 countries, including the United States, France, United Kingdom, Germany, and Canada. We can see that so far Airbnb is still more popular in North America and Europe than other areas in the world.

Except the reason that Airbnb is a US-based company, there may be other reasons for such user composition. In this work, we pick the top 8 countries with most Airbnb users, and try to discover the correlation between the number of Airbnb users and the social and economic factors like GDP, GDP per capita



(a) Home Country Distribution (b) Percentage of Different Types (c) Number of Reviews

Fig. 1: Analysis of User Profiles and Reviews

and population. In Table I, the figures of GDP and GDP per capita were retrieved from the website of the International Monetary Fund (IMF)[2], while the population information was obtained from the Department of Economic and Social Affairs of the United Nations[3].

We model each column of Table I as a vector, and denote the vectors by v_{user_num}, v_{GDP}, $v_{GDP\_per\_capita}$, and $v_{population}$, respectively. After that, we calculate the Pearson correlation coefficients between v_{user_num} and each of the other three vectors. The resulted correlation coefficients are denoted as r_{GDP}, $r_{GDP\_per\_capita}$, and $r_{population}$, respectively. The value of the correlation coefficient reflects the impact of the corresponding vector on the number of registered Airbnb users in each country. Note that a Pearson correlation coefficient is between -1 and 1. 1 refers to total positive linear correlation, 0 indicates no linear correlation, and -1 refers to total negative linear correlation. Concerning only the top 8 countries with most Airbnb users, r_{GDP} is 0.9927, $r_{GDP\_per\_capita}$ is 0.6124, and $r_{population}$ is 0.9873. If we extend the scope to include all the Airbnb users around the world, the values of r_{GDP}, $r_{GDP\_per\_capita}$ and $r_{population}$ are 0.8654, 0.2950, and 0.2049, respectively. Obviously, the number of registered users in a certain country is positively relevant to this country's GDP, whereas it is less relevant to the GDP per capita and the population.

Verified IDs As a method of improving the trust between users, Airbnb encourages users to submit their online and offline IDs for verification. After a user adds her ID information to her personal profile, Airbnb is responsible for verifying that the user does own the ID in question. From the values of the "Verified ID" field, we can find out which types of IDs have been verified. According to the personal profiles we have collected, most users have chosen to verify their "E-mail address", "Phone Number", and "Facebook Account". As shown in Fig. 1(b), these three types cover 52.82%, 35.64% and 26.29% of all the verified IDs, respectively. These are followed by the government-issued offline IDs, such as Passport and Driver License, which takes 16.65%. A user can request to become a "verified user". Upon request, Airbnb will verify the following items, including an online ID, a government-issued offline ID, a profile photo, a phone number, and an email address. Only 19.43% of all users are verified.

Reviews Airbnb users can write reviews for their hosts or guests. Fig. 1(c) demonstrates the distribution of the number of published reviews per user. It fits nicely with the power law model, i.e., $P(k) \propto k^{-\alpha}$ [5]. To evaluate how well the model fits the distribution, we adopt the coefficient of determination, i.e., the R^2 value. The value of R^2 ranges from 0 to 1. The larger the value is, the better the fitting is. When α is set to 2.4468, the value of R^2 is 0.9557, indicating a nice fit with the distribution of the number of per-user reviews.

Among all the users who have posted at least one review, only 3.53% of them have played both guest and host roles, and 90.76% of them only act as guests. Compared with the 5.71% of users who have written reviews as hosts, the number of guests is much bigger, which means that most of people use Airbnb for searching and booking accommodation instead of leasing out their apartments.

Superhost An Airbnb usr can become a "superhost" and get a superhost badge on her profile page, if she satisfies certain requirements, including hosting at least 10 groups of guests, receiving a "5-star" for at least 80% of the reviews posted by her guests, and completing each of the confirmed reservations. According to our study, there are only 68,883 superhosts, which means about 0.16% of Airbnb users are classified as superhosts.

About Me Besides the above-mentioned fields, there is an "About Me" field in each user's profile. It allows a user to add more information about herself. Optionally, users could add their "School", "Work" and "Language" information. Among all users, 28.66% have provided the "School" information, 10.16% have added the "Work" information, and 9.36% have said something about their "Language".

2) Temporal Evolution of Airbnb Demographic: According to the registration time of each Airbnb user, we can review the growth of Airbnb in terms of the number of registered users in the past 8 years. As illustrated in Fig. 2(a), both the number of registered users and the amount of published reviews have been growing steadily. The figures grow much faster during summers, showing that people are more active in traveling in summer.

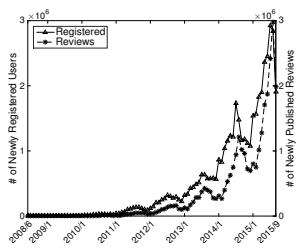
We further look into the geographical distribution of Airbnb users and measure the diversity of home countries. Here we

[2]http://www.imf.org/

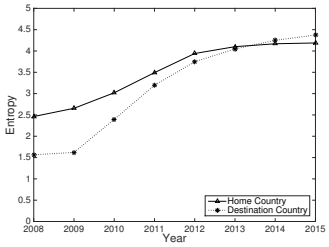
[3]https://esa.un.org/unpd/wpp/

TABLE I: Number of Registered Users v.s. GDP / GDP per Capita / Population

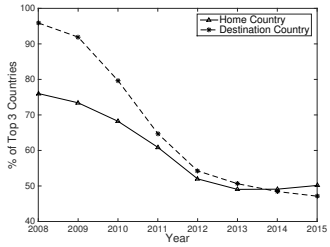
Country	Number of registered users	GDP (millions of USD)	GDP per capita (USD)	Population
United States	12,979,691	18,561,930	56,084	324,119,000
France	2,910,159	2,488,280	37,653	64,668,000
United Kingdom	2,195,446	2,649,890	43,902	65,111,000
Germany	1,834,505	3,494,900	40,952	80,682,700
Canada	1,528,011	1,532,340	43,413	36,286,200
Italy	1,507,997	1,852,500	29,867	59,801,000
Australia	1,407,956	1,256,640	51,181	24,309,000
Spain	1,215,428	1,252,160	25,843	46,065,000



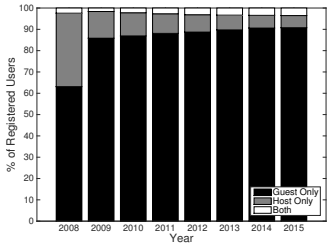
(a) Number of Registered Users and Published Reviews



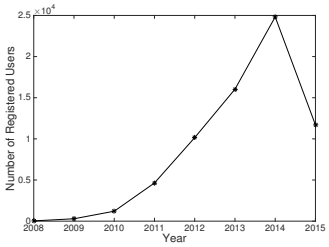
(b) Home Country Entropy



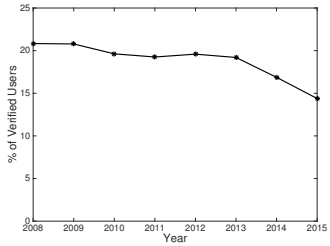
(c) Percentage of Top 3 Home/Destination Countries



(d) Distribution of Hosts/Guests/Both



(e) Number of Superhosts



(f) Percentage of Verified Users

Fig. 2: Demographic Analysis (Temporal)

introduce a metric called "home country entropy" and denote it by E_{home}. E_{home} can be calculated using the formula $E_{home} = -\sum_{i=1}^{k} p_i log_2 p_i$, where p_i refers to the fraction of users coming from the $i-th$ country. The value of E_{home} increases with the diversity of home countries. Similarly, we calculate the "destination country entropy" (E_{dest}). The information of "destination country" can be extracted from the reviews made for each trip. According to Fig. 2(b), Airbnb has become more and more globally recognized, in terms of both users' home countries and destination countries.

To verify the results of the home/destination country entropy, we check the distribution of the most popular home countries and destinations. Fig. 2(c) shows how many percentage of trips are made by the users from the top 3 most popular home countries and how many percentage of trips are made to the top 3 most popular destinations. We can see that in the first 3 years the percentages are larger than 60%. The curves started to drop in 2011, and got stabilized around 50%. These results are consistent with the values of the home/destination country entropy. In short, Airbnb is growing not only the number of registered users, but also its geographic diversity.

Regarding the number of hosts and guests, as shown in Fig. 2(d), the proportion of pure guests among all Airbnb users has grown from 63.11% to 90.76%. Meanwhile, the proportion of pure hosts keeps decreasing, while more and more users would play both the roles of hosts and guests. Compared with the growing number of travelers, the number of listed properties is growing relatively slow. If we look at the number of superhosts, users have joined the "superhost" group from time to time. Fig. 2(e) shows how many users registered in a certain year have become superhosts by end of 2015. Compared with hosts registered before 2015, fewer hosts registered after 2015 have become superhosts. This is partly due to the strict requirements of becoming superhosts, for example, superhosts must have hosted at least 10 trips.

Although the number of Airbnb users is growing steadily, the proportion of verified users has not grown. According to Fig. 2(f), among all the Airbnb users, around 20% of them are verified users. This number is smaller for those registered in 2014 and 2015. We believe more of them will apply for verification in the future.

B. Social Interaction Analysis

We utilize the review graph generated from the collected personal profiles for analyzing the social interactions between Airbnb users. We will first measure the complete review graph using the graph metrics listed below, and then analyze the

temporal evolution of the review graph and the characteristics of verified users.

- **Indegree and outdegree**: Indegree refers to the number of incoming edges a node has. The indegree of a node (user) is equal to the number of visitors the user has hosted. Outdegree refers to the number of outgoing edges a node has. The outdegree of a node (user) indicates the number of users she has visited.
- **PageRank**: PageRank is a metric that measures and ranks the importance of nodes in a graph [17]. It has been used by Google to rank the websites. We use this metric to discover "important users" in the graph.
- **Strongly connected component (SCC)**: An SCC is a subgraph where there is a path between any two nodes, while no additional node or edge can be added to this subgraph without breaking the nature of "strongly connected".
- **Weakly connected component (WCC)**: A WCC is a subgraph where there is a path between any two nodes when all edges are viewed as undirected. In addition, no additional node or edge can be added to this subgraph without breaking the nature of "weakly connected".
- **Communities**: A social network often exhibits a community structure. A community is formed by a number of nodes which are densely connected internally.

1) Review Graph: Static Analysis: ***Indegree and Outdegree*** The Cumulative Distribution Function (CDF) of indegree and outdegree among all the nodes is illustrated in Fig. 3(a) and Fig. 3(b). For comparison, we also visualize the CDF of indegree and outdegree among verified users and superhosts. The indegree and outdegree of the nodes corresponding to verified users and superhosts are relatively high, compared with other nodes. According to [16], the median indegree and outdegree of Twitter social graph are 16 and 39, respectively. Obviously, the numbers are much smaller in case of Airbnb, which means the review graph of Airbnb is rather sparse.

PageRank We use PageRank to measure the importance of each node in the review graph. We choose 1000 nodes with the largest PageRank values and compare their characteristics with those of the entire Airbnb population. Among these 1000 nodes, 31.1% of them are pure hosts, while 68.9% of them play both roles. None of these 1000 nodes is purely a guest. We can see that hosting more is a critical indicator for becoming an important node in the review graph. In addition, the median indegree and outdegree of these 1000 nodes are 732 and 3, respectively. Both figures are much higher than those of the entire review graph.

Regarding the verification status, 100%, 89%, and 79.3% of the top 10, 100, 1000 nodes with highest PageRank values are verified users. Although verified users only cover 19.43% of the nodes in the entire G, verified users are more likely with higher PageRank values. Also, we are aware that about 14.48% of the top 1000 nodes are multi-user accounts, for example, the user name is "Alice and Bob" or "Carol & Tom". In contrast, only 0.282% of Airbnb accounts are multi-user accounts. Therefore, a viable portion of most important Airbnb accounts are operated by multiple people, for example, a couple or a family.

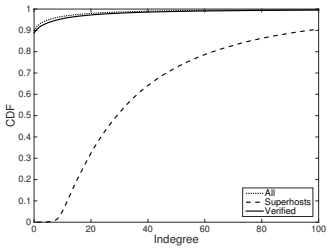
TABLE II: Percentage of Users in Top 3 Countries per Community

Community	Countries (% of Users)		
$C1$	US (66.54%)	CA (7.18%)	UK (3.26%)
$C2$	US (76.76%)	CA (3.09%)	UK (2.65%)
$C3$	FR (31.95%)	US (10.41%)	ES (8.21%)
$C4$	US (16.55%)	IT (14.92%)	FR (14.01%)
$C5$	AU (28.76%)	US (12.54%)	CN (5.56%)
$C6$	FR (22.78%)	ES (13.33%)	US (10.76%)
$C7$	DE (16.80%)	US (12.80%)	FR (10.61%)
$C8$	UK (36.81%)	US (13.16%)	FR (8.53%)
$C9$	US (54.41%)	FR (5.59%)	DE (4.83%)
$C10$	US (15.72%)	DE (13.98%)	FR (9.80%)

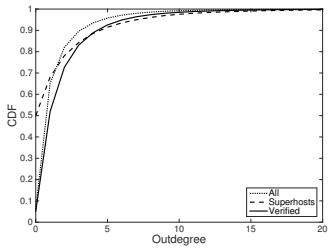
SCC and WCC We are interested in the connectivity among users in G. The sizes of the five largest SCCs are 63497, 13, 5, 4, 3, respectively. The largest SCC only covers 0.33% of all nodes, and the second largest SCC has only 13 nodes. This means very few nodes are strongly connected with each other. Differently, the sizes of the top five largest WCCs are 10969215, 15, 15, 15, and 14, respectively. We can see that the largest WCC covers 98.28% of nodes in G. Different from the small sizes of the SCCs, there is one giant WCC covering the major portion of all Airbnb users. In other words, most of the Airbnb users are weakly connected.

Communities The concept of community structure is widely used to study complex networks. If the network has a "community structure", the nodes can be split into different communities. Nodes from the same community are densely connected with each other, while nodes from different communities are sparsely connected. To study the communities in the Airbnb network, we adopt the widely used Louvain algorithm [2]. This algorithm is initially designed for undirected graphs. Following the practices in [11], we convert the review graph into an undirected graph by simply considering each edge as undirected. Louvain algorithm can assign each node of the network to one and only one community. It optimizes a metric known as "modularity". The value of modularity is between -1 and 1. Normally, if this value is larger than 0.4 [7], we can conclude that the network has a significant community structure. For G, the corresponding modularity value is 0.66, which means that the Airbnb network has a viable community structure. Also, our results show that there are 81308 communities among all nodes in G. Fig. 3(c) shows that sizes of the largest 30 communities. Among all communities, top 10 of them have covered 44.19% of nodes in G, and top 30 of them have covered 56.99% of nodes in G. In particular, the country composition of the top 10 communities are shown in Table II. We find that each of these communities has only one or very few dominant countries.

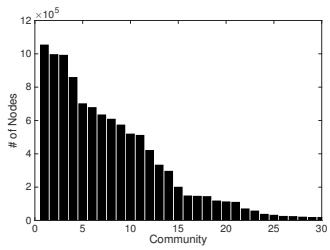
2) Temporal Evolution of the Review Graph: We are not only interested in the up-to-date structure of the review graph, but also how this graph has been constructed gradually. In this subsection, we study the temporal evolution of the review graph, taking the creation time of each node and edge into



(a) Indegree

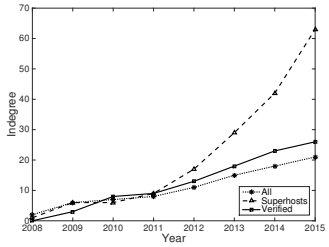


(b) Outdegree

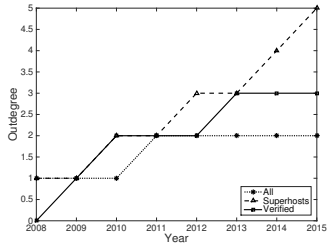


(c) Community Size Distribution

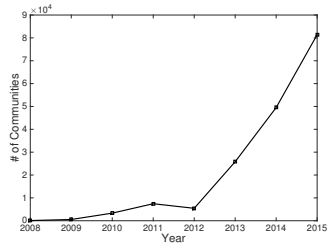
Fig. 3: Static Analysis of the Review Graph



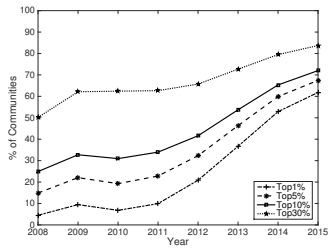
(a) Indegree



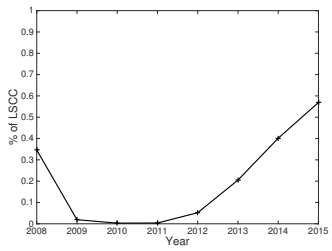
(b) Outdegree



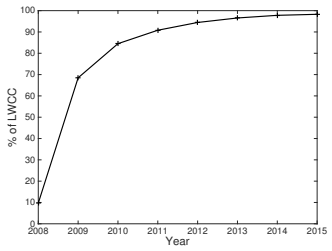
(c) # of Communities



(d) Top Communities



(e) LSCC



(f) LWCC

Fig. 4: Dynamic Analysis of the Review Graph

account. According to Fig. 4(a) and Fig. 4(b), the average indegree and outdegree of nodes in G grow steadily, as the platform is developing rapidly. In 2008, the 80th percentile values of the indegree and outdegree are 2 and 1, respectively. In 2015, these two values become 21 and 2. Consequently, the review graph becomes denser and denser. More and more people are linked with each other through Airbnb. In Fig. 4(c), we can see the number of communities also grows year by year. Meanwhile, as shown in Fig. 4(d), the fractions of nodes within the top 1%, 5%, 10% and 30% are becoming larger and larger. We also pay attention to the fraction of the largest strongly connected component (LSCC) and the largest weakly connected component (LWCC) among all nodes in G. For the LSCC (Fig. 4(e)), we can see it decreases for the first few years, and grows since 2011. However, the percentage of the LSCC is very small (less than 0.4%) all the time. Differently, we can see the percentage of LWCC (Fig. 4(f)) increases year to year. In 2008 about 10% users belong to the LWCC. This number increases yearly. Finally, in 2015, more than 90% users are involved in the LWCC. We believe that now most of the users are weakly connected.

C. Mobility of Airbnb Users

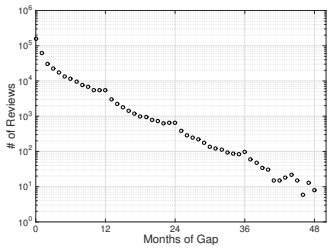
1) Spatial-Temporal Analysis: Understanding the spatial-temporal characteristics is important for an online lodging service. Thanks to the near real-time nature of review publishing, we can infer the users' mobility patterns by referring to published reviews.

We first explore the distribution of the time gap between two successive reviews published by the same user in Fig. 5(a), on a monthly base. We can see that when the time gap becomes larger, the number of corresponding successive review pairs become fewer. However, if the gap value can be divided by 12 months, there is a viable "peak". It shows that some travelers undertake their travels on a yearly base.

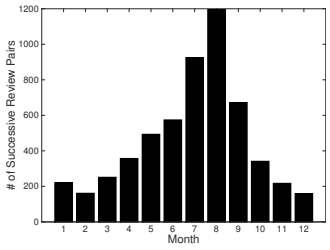
We further study the case with a time gap of one year, and categorize these review pairs according to the published month of the first review of them in Fig. 5(b). The x-axis denotes the published month, and the y-axis shows the number of successive review pairs with a time gap of one year. We can see most of the yearly travels take place in July and August. In Fig. 5(c), we can see the average time gap of successive review pairs of the users coming from the top 10 countries.

TABLE III: Fraction Distribution of "Home - Destination" County Pairs

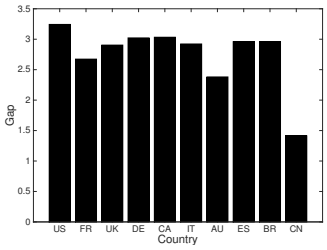
Home \ Dest.	US	FR	UK	DE	CA	IT	AU	ES	BR	CN	$\sum_j H_{ij}$
US	0.3958*	0.0221*	0.0170	0.0082	0.0187	0.0236*	0.0044	0.0134	0.0029	0.0013	0.5073
FR	0.0161	0.0268*	0.0083	0.0041	0.0030	0.0127	0.0013	0.0097	0.0009	0.0002	0.0831
UK	0.0205*	0.0130	0.0445*	0.0056	0.0024	0.0109	0.0036	0.0099	0.0009	0.0003	0.1116
DE	0.0173	0.0078	0.0066	0.0167	0.0021	0.0091	0.0021	0.0077	0.0007	0.0002	0.0703
CA	0.0230*	0.0050	0.0036	0.0016	0.0293*	0.0050	0.0011	0.0032	0.0004	0.0002	0.0725
IT	0.0047	0.0042	0.0033	0.0021	0.0003	0.0087	0.0004	0.0031	0.0002	0.0001	0.0271
AU	0.0160	0.0072	0.0070	0.0026	0.0019	0.0072	0.0343*	0.0035	0.0005	0.0002	0.0805
ES	0.0039	0.0031	0.0028	0.0017	0.0003	0.0026	0.0002	0.0062	0.0002	0.0001	0.0210
BR	0.0037	0.0015	0.0009	0.0006	0.0005	0.0011	0.0001	0.0006	0.0030	0.0000	0.0121
CN	0.0048	0.0014	0.0012	0.0006	0.0004	0.0013	0.0009	0.0005	0.0000	0.0032	0.0144
$\sum_i H_{ij}$	0.5059	0.0921	0.0951	0.0437	0.0591	0.0823	0.0485	0.0578	0.0096	0.0058	1.0000



(a) Distribution of the Time Gap

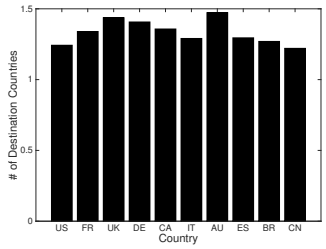


(b) Gap: 12 Months

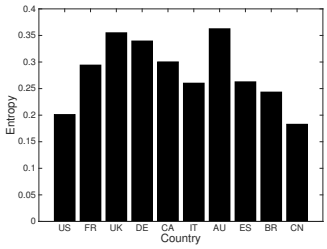


(c) Avg. Time Gaps of Different Countries

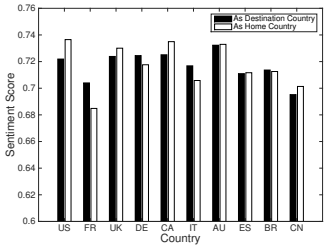
Fig. 5: Distribution of the Time Gap Between Two Successive Reviews



(a) Avg. Number of Visited Countries



(b) Entropy



(c) Sentiment Analysis

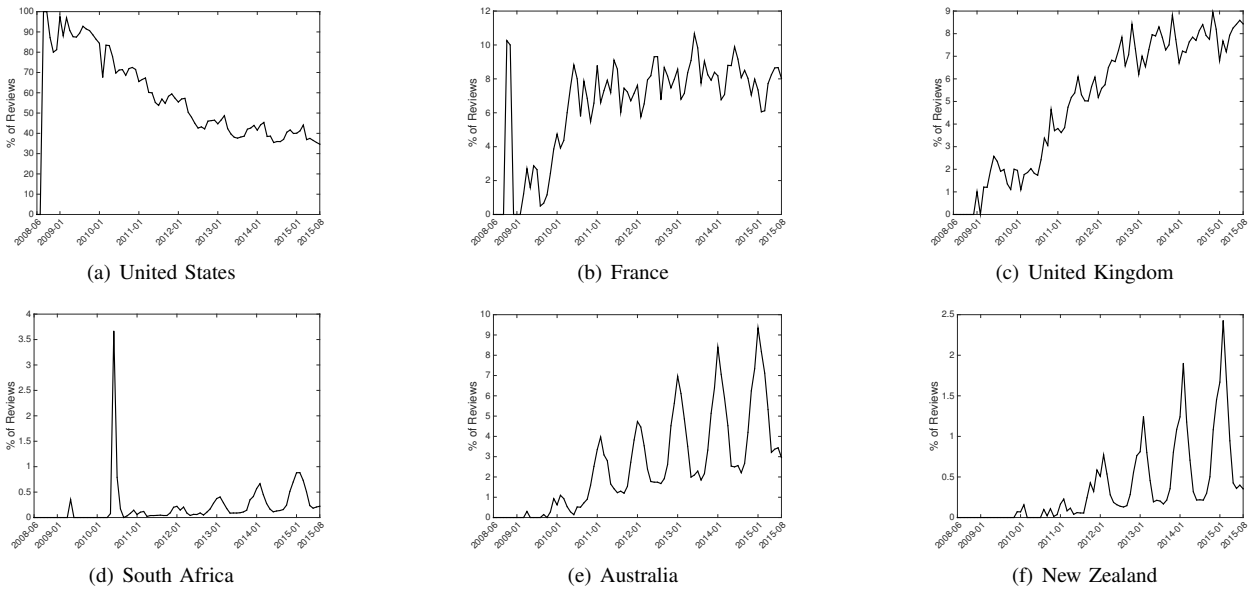
Fig. 6: Global Mobility of Users From Different Countries

We find that users from China has the smallest average time gap, while the users from the United States has the largest average time gap.

For each review, we can extract the home country of the publisher. Meanwhile, we know the country that she visits, which is known as the "destination country". Therefore, each review has a corresponding "home-destination country pair". In Table III, we use a matrix H to quantify the fraction distribution of home-destination country pairs. For simplicity, we only consider the users coming from the top 10 countries. We select elements with a value more than 0.02 and mark them with "*". In this matrix, we can see that most of the users have paid more visits to their home countries. In terms of the number of reviews, the three most popular destination countries are United States, United Kingdom and France.

In Fig. 6, we can see the global mobility of users from top 10 countries. We use two metrics, i.e., the number of visited countries, and the destination country entropy. The first metric can simply count the number of destination countries a user have visited. From Fig. 6(a), we can see that users from Australia and United Kingdom have visited more countries. From Fig. 6(b), we show the diversity of visited countries by calculating the entropy of destination countries. Similarly, we can see a higher diversity of destination countries for users coming from Australia and United Kingdom. We also calculate the average sentiment score of each home country and destination country, and show the results of the top 10 countries in Fig. 6(c). We find that there is very little difference among these countries. In average, users from the United States are slightly happier. Meanwhile, as a destination, Australia can make more people happy.

To understand where the users go from a temporal aspect, we also conduct country-level analysis from a destination country's perspective. We can see the evolution of the visitor population over time, and we have examined the top 30 countries according to the user population. Due to the page limit, we pick six representative countries for our study. On one hand, we select the United States, France, and United



(a) United States (b) France (c) United Kingdom

(d) South Africa (e) Australia (f) New Zealand

Fig. 7: Temporal Behavior of Different Destination Countries

Kingdom, as they have the largest user population. Since these three countries are in the Northern Hemisphere, we select three countries in the Southern Hemisphere, i.e., South Africa, Australia and New Zealand. The results are shown in Fig. 7. For each country, we show the user popularity from a temporal view. The x-axis denotes the time information, and the y-axis represents the percentage of reviews a destination country has received in a certain month. We can see all the six countries have shown a seasonal periodicity. One significant difference between countries in the two hemisphere is the peak period of a year. In the countries of the Southern Hemisphere, there is always a peak in January and a valley in July. Differently, we observe an almost opposite trend for countries of the Northern Hemisphere. As we mentioned earlier, Airbnb is widely spread around the world; accordingly we can see the share of the reviews of United States-based apartments is going down. Another interesting finding is about South Africa, there is a significant and unusual peak in June and July of 2010. We believe that this is because the FIFA World Cup 2010, which has attracted numerous soccer fans from around the world.

2) Prediction: In this subsection, we investigate the predictability of user movements. In particular, we are interested in whether a user will travel aboard. In our study, we focus on users who have conducted at least one trip in 2015. Moreover, we exclude the users who have completed less than 7 trips on Airbnb, since they do not have enough historical data for the prediction. Among the rest of users, we group them into two categories, i.e., users whose latest trip is an international trip, and users whose latest trip is a domestic trip. We call the first group users "international users", and the second group of users "domestic users". We randomly pick 24,000 international users and 24,000 domestic users to form a training dataset.

We select a number of key features to distinguish between these two types of users. These features belong to four categories: (1) the ratio between international and domestic trips; (2) the time interval between each two successive trips; (3) the number of trips within a certain time interval; (4) the demographic information. Given a training set and the selected features, we apply different supervised machine learning algorithms to predict whether a user is an international user. The algorithms we test include XGBoost [4], C4.5 decision tree [19], Random Forest [3], Naive Bayes [14] and supporting vector machine (SVM) [10]. We use 10-fold cross-validation to test the classification accuracy of these algorithms. Three classic metrics are introduced, i.e., precision, recall, and F1-score. Precision means the fraction of identified international users who have really traveled abroad for their latest trips. Recall indicates the fraction of international users who have been accurately detected. F1-score represents the harmonic mean of precision and recall. According to Table IV, the XGBoost algorithm outperforms other algorithms and the overall F1-score is as high as 0.766. Therefore, the selected features could accurately distinguish international users from domestic users. To evaluate the importance of each feature, we use χ^2 (Chi square) statistic to measure each feature's discriminative power [21]. The results are shown in Table V.

IV. Related Work

Analysis of online service users always starts with the collection of user data. A straightforward way is to obtain all the data directly from the back-end servers. For example, Zhao et al. [22] have explored the evolution of the Renren network using the data obtained from the back-end. However, very few online service providers have opened their data for public research. Furthermore, many of them have applied mechanisms such as per-IP address rate limit to prevent large-scale data crawling. As in [6], we apply a distributed data

TABLE IV: Prediction of "International Users"

Algorithm	Parameter	Precision	Recall	F1-Score
XGBoost	learning rate=0.09, max_depth = 6, gamma = 0.2, seed = 2	0.792	0.741	0.766
Random Forest	247 trees,depth=0	0.774	0.737	0.755
C4.5(J48)	Instance/leaf M=1,Confidence factor C=0.006	0.779	0.735	0.756
BayesNet	4 children,4 parents	0.782	0.726	0.753

TABLE V: χ^2 statistic

Rank	Feature	χ^2
1	Fraction of International Trips	15227.842064
2	Fraction of International Trips in 2015	12138.063148
3	Number of International Trips	11985.024126
4	Whether the 2nd Latest Trip is International	10841.01484
5	Home Country's GDP	9103.609107

crawling approach to collect all the personal profiles of Airbnb users, which allows us to conduct a comprehensive analysis.

Quattrone et al. [18] have crawled the Airbnb data of the city of London, and have studied the problem of regulating Airbnb. Their work investigated Airbnb from a socio-economic angle, and conducted a series of temporal-spatial analysis of Airbnb properties and demands in London. Ma et al. [15] focused on the Airbnb hosts, and studied how hosts describe themselves in their profile pages. Their study was based on 67,465 hosts coming from 12 cities in the United States. Differently, our work focuses on the interactions between users, and have extended the scope to the entire set of Airbnb users.

Conventionally, a "social graph" models a number of users and the "friendship" connections among them. The connection between users does not necessarily reflect the real interactions between them. To solve this issue, Wilson et al. [20] proposed to describe the interactions with an "interaction graph", and demonstrated through a data-driven study that the interaction graph can describe the user activities more efficiency than the social graph relying on social links only. Jiang et al. further studied latent interactions in the Renren social network [13] based on the profile visit histories. Their study also demonstrated that latent interactions are more meaningful than social links. Similarly, we construct our review graph based on user interactions. Our review graph models the user mobility and interactions on a global scale.

V. Conclusions

In this paper, we conduct a comprehensive user behavior analysis of Airbnb, a leading online lodging service. Our study covers different aspects, including the user composition, the interactions between users, and the cross-country mobility patterns of the users. To the best of our knowledge, our study presents the first comprehensive and evolutionary analysis of Airbnb users on a global scale. In the future, we plan to analyze the Airbnb users' online behavior and offline activities as an integrated whole. Also, we aim to detect the spam accounts using deep learning technologies [9].

Acknowledgement

This work is sponsored by National Natural Science Foundation of China (No. 61602122, No. 71731004), Natural Science Foundation of Shanghai (No. 16ZR1402200), Shanghai Pujiang Program (No. 16PJ1400700), EU FP7 IRSES MobileCloud project (No. 612212) and Lindemann Foundation (No. 12-2016). Yang Chen is the corresponding author.

References

[1] S. Bird. NLTK: The Natural Language Toolkit. In *Proc. of COLING/ACL*, 2006.
[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.
[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
[4] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proc. of ACM KDD*, 2016.
[5] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703, 2009.
[6] C. Ding, Y. Chen, and X. Fu. Crowd Crawling: Towards Collaborative Data Collection for Large-scale Online Social Networks. In *Proc. of ACM COSN*, 2013.
[7] S. Fortunato and M. Barthlemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104:36–41, 2007.
[8] A. Fradkin, E. Grewal, D. Holtz, and M. Pearson. Bias and Reciprocity in Online Reviews: Evidence From Field Experiments on Airbnb. In *Proc. of ACM EC*, 2015.
[9] Q. Gong, Y. Chen, X. He, Z. Zhuang, T. Wang, H. Huang, X. Wang, and X. Fu. DeepScan: Exploiting Deep Learning for Malicious Account Detection in Location-Based Social Networks. *IEEE Communications Magazine*, 2018.
[10] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.
[11] D. Hric, R. K. Darst, and S. Fortunato. Community detection in networks: Structural communities versus ground truth. *Phys. Rev. E*, 90:062805, Dec 2014.
[12] C. J. Hutto and E. Gilbert. VADER: A parsimonious rule-based model for sentiment analysis of social media text. In *Proc. of AAAI ICWSM*, 2014.
[13] J. Jiang, C. Wilson, and et al. Understanding Latent Interactions in Online Social Networks. In *Proc. of ACM IMC*, 2010.
[14] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proc. of UAI*, 1995.
[15] X. Ma, J. Hancock, K. L. Mingjie, and M. Naaman. Self-disclosure and Perceived Trustworthiness of Airbnb Host Profiles. In *Proc. of ACM CSCW*, 2017.
[16] S. A. Myers, A. Sharma, P. Gupta, and J. Lin. Information Network or Social Network?: The Structure of the Twitter Follow Graph. In *Proc. of WWW '14 Companion*, 2014.
[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, November 1999.
[18] G. Quattrone, D. Proserpio, and et al. Who Benefits from the "Sharing" Economy of Airbnb? In *Proc. of WWW*, 2016.
[19] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
[20] C. Wilson, B. Boe, and et al. User Interactions in Social Networks and Their Implications. In *Proc. of ACM EuroSys*, 2009.
[21] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of ICML*, 1997.
[22] X. Zhao, A. Sala, C. Wilson, X. Wang, S. Gaito, H. Zheng, and B. Y. Zhao. Multi-scale Dynamics in a Massive Online Social Network. In *Proc. of ACM IMC*, 2012.

Wrinkles in Time: Detecting Internet-wide Events via NTP

Meenakshi Syamkumar*, Sathiya Kumaran Mani*, Ramakrishnan Durairajan†, Paul Barford*‡ and Joel Sommers§
*University of Wisconsin-Madison, †University of Oregon, ‡comScore, Inc, §Colgate University

***Abstract*—Understanding the nature and characteristics of Internet events such as route changes and outages can serve as the starting point for improvements in network configurations, management and monitoring practices. However, the scale, diversity, and dynamics of network infrastructure makes event detection and analysis challenging. In this paper, we describe a new approach to Internet event measurement, identification and analysis that provides a broad and detailed perspective without the need for new or dedicated infrastructure or additional network traffic. Our approach is based on analyzing data that is readily available from Network Time Protocol (NTP) servers. NTP is one of the few on-by-default services on clients, thus NTP servers have a broad perspective on Internet behavior. We develop a tool for analyzing NTP traces called *Tezzeract*, which applies Robust Principal Components Analysis to detect Internet events. We demonstrate Tezzeract's efficacy by conducting controlled experiments and by applying it to data collected over a period of 3 months from 19 NTP servers. We also compare and contrast Tezzeract's perspective with reported outages and events identified through active probing. We find that while there is commonality across methods, NTP-based monitoring provides a unique perspective that complements prior methods.**

I. Introduction

Unexpected events in the Internet can impact users in a variety of ways. On one end of the spectrum are small-scale events such as localized route changes that occur for any number of reasons and that cause only a brief increase in latency for users. At the other end of the spectrum are large-scale events such as outages (*e.g.,* the Baltimore Howard Street Tunnel fire [1]) that can cause wide-spread service disruptions and impact many users for many days.

Understanding the characteristics of unexpected events such as their prevalence and impact is important for planning, configuration and management of networks. It can illuminate weaknesses and vulnerabilities in network design and implementation. It can also clarify how measurement, monitoring and diagnostic capabilities might be deployed more effectively and efficiently. Furthermore, real-time event detection offers the opportunity to identify the scope and details of events and restore service in a timely fashion.

There are a number of challenges in developing the capability to identify and understand network events. First is the problem of gathering measurements that can provide a sufficient reach and detail in an efficient and timely fashion. Second is the problem of detecting and localizing the scope of events within a potentially vast amount of measurement data in an accurate and reliable fashion. Third is the problem of event diagnosis that can lead to effective and efficient remediation. Prior methods for Internet event detection (*e.g.,* [2]–[5]) typically rely on data from a dedicated measurement infrastructure, and can suffer from noise that is inherent in data collected in the Internet.

In this paper, we address the problem of *Internet event detection*. We define an "Internet event" as a sudden change in conditions that manifests as a change in packet latency experienced by a cluster of clients. The goal of our work is to develop a method for Internet event detection that can provide both a broad and detailed perspective without the need for deployment of new or dedicated measurement infrastructure. Several prior studies have utilized *existing* wide-area infrastructure for path failure monitoring and latency change detection (*e.g.,* [6], [7]). We adopt a similar approach, however real-time monitoring is beyond the scope of our current work.

Our method for Internet event detection utilizes a new source of data: measurements from Network Time Protocol (NTP) servers. NTP is one of the few on-by-default services on clients and it is ubiquitously deployed, thus NTP data can provide a broad perspective on Internet state. The efficacy of extracting latency measurements from NTP data and the diverse coverage of clients provided by NTP servers was demonstrated in [8]. Our study utilizes an expanded technique to extract one way delays (OWDs) between NTP clients and servers [9].

We develop a technique for detecting Internet events from OWDs extracted from NTP data. Intuitively, a jump in OWDs between a cluster of clients and a server is an indication of an event. Our technique is based on applying Robust Principal Component Analysis (RPCA) [10] to OWDs for client aggregates (defined by network prefixes) *ex post facto*. Events are identified when specified thresholds are exceeded as explained in §III. This approach enables the scope, duration and other details of events to be identified. RPCA is attractive for our application since it is more resilient to noisy data than standard PCA [11].

Our RPCA-based event detection technique is realized in a tool we call *Tezzeract*. We conduct a sensitivity analysis to establish a configuration for Tezzeract that will provide consistent and reliable results. Next, we demonstrate Tezzeract's ability to detect events through a set of controlled experiments, by injecting randomized events into NTP traces. Following that, we report on the results of events identified in ~1B NTP transactions collected over a period of 3 months from 19 NTP servers in the US. We find that the average number of events detected per day varies by NTP server and the size of its client population. We also find that the number of events detected per day varies based on whether client-to-server (c2s) or server-to-

ISBN 978-3-903176-08-9 © 2018 IFIP

client (s2c) OWDs are considered. Finally, we examine events that are detected by multiple servers, and find that as many as 10 out of the 19 traces may show an event simultaneously.

We compare and contrast the events identified by Tezzeract with two other data sources including *(i)* the ongoing Internet-wide Census and Survey project at ISI [12] and *(ii)* public reports of actual outage events. The comparisons are not intended to "validate" our method since no ground truth for Internet-wide events is available (an exception being reported events). Rather they are meant to demonstrate the utility of our method and how it compares and contrasts with other detection methods. In the case of the comparison with ISI's active probe data, we find that Tezzeract is more conservative, reporting many fewer events per day. This can be attributed to the liberal definition of event used in [12], which is simply missing ping measurements. However, Tezzeract does detect up to 67% of the events identified in the ISI data. Finally, comparison with a reported outage shows that Tezzeract effectively identifies the event and the underlying OWDs used to drive the analysis provide a useful perspective on the event's impact.

In summary, this paper makes the following contributions. *(1)* We introduce the use of NTP traces for Internet event detection, which enable broad and detailed analysis without the need for dedicated measurement infrastructure; *(2)* We describe a new method for Internet event detection based on applying RPCA to latency measurements from client clusters, which is implemented in a tool called Tezzeract; *(3)* We demonstrate the efficacy of NTP traces and our method by reporting on controlled laboratory experiments, applying Tezzeract to a large NTP data corpus, and comparing and contrasting with events detected by other methods.

We find that while there is commonality across methods, NTP-based monitoring provides a perspective that is unique, accurate and complements prior methods.

II. Datasets

A. NTP data

The NTP is both a protocol and a global hierarchy of reference servers. At the top of the server hierarchy, referred to as stratum 0, are high-precision time sources such as GPS-based and atomic clocks. These servers act as highly accurate references for servers in the next level of the hierarchy, stratum 1, which are also known as *primary* servers. *Secondary*, stratum 2 servers synchronize from stratum 1 servers and so forth down to stratum 15, which is the lowest level of the hierarchy. For redundancy, servers may also peer with others at the same level.

NTP clients compute a precise time estimate by synchronizing with one or more than one servers. Hosts running a commodity operating system are typically configured to synchronize with a default NTP server(s) (*e.g.*, `time.windows.com`, `time.apple.com`, `0.pool.ntp.org`), but can be configured to use a specific NTP server or set of servers. NTP hosts or clients typically connect to reference clocks that are stratum 2 or higher. Lists of stratum 1 and stratum 2 servers are maintained by `ntp.org`. Synchronization from these servers typically requires permission from the server administrators.

We assembled the dataset used in our study from NTP servers that are listed as part of `pool.ntp.org`. We started by reaching out to several NTP operators and explained our research goals; several operators responded positively. Out of the many who responded, we carefully selected eight NTP operators who maintain 19 different servers and obtained datasets in the form of full packet (libpcap) traces.

An intrinsic component of NTP (and in turn in the traffic captured at the servers) is the presence of timestamps in packets that are exchanged between NTP clients and servers. In particular, four timestamps are included in the NTP packets that are exchanged as part of the NTP synchronization procedure (known as *polling*): $t0$, the time at which a clock synchronization request is sent; $t1$, the time at which the request is received at the NTP server; $t2$, the time at which the response is sent by the server; and finally $t3$, the time at which the response is received by the client. We use these four timestamps to calculate the client-to-server (c2s) and server-to-client (s2c) one-way delays (OWDs). The NTP protocol running on clients determine the *polling interval* (in seconds), which is the period between NTP packets sent to a server.

Unfortunately, the captured packets have no explicit information about the level of synchronization of client(s) with NTP server(s). As a result, we must identify and remove packet exchanges between clients and servers in which the clients are observed not to be in synchronization (otherwise OWD estimates would be inaccurate). We utilize a filtering method described in [9] that employs NTP-specific heuristics on extracted OWD values, polling intervals and NTP packet fields, and divides clients into various *precision tiers* based on inferred synchronization quality. We use the OWDs from only the highest precision tier *i.e.,* clients that exhibit tight synchronization with NTP servers.

TABLE I
Summary of NTP traces used in this study.

Server ID	Server Organization	Total Measurements	Total Unique Clients	Client Prefixes [Fraction]
AG1	Independent	36,309,416	171,326	19,633 [7.6e-04]
CI1	ISP	1,483,460	549	158 [5.4e-06]
CI2	ISP	780,580	342	145 [3.5e-06]
CI3	ISP	1,305,499	357	173 [3.2e-06]
CI4	ISP	665,732	240	96 [3.5e-06]
EN1	ISP	727,873	260	140 [3.8e-06]
EN2	ISP	813,531	229	106 [3.5e-06]
JW1	Commercial	2,394,120	3,318	1,377 [4.5e-05]
JW2	Commercial	2,914,157	3,874	1,567 [3.8e-05]
MW1	University	1,441,746	10,232	33 [2.8e-05]
MW2	University	40,129,376	49,179	18,369 [1.1e-03]
MW3	University	8,514,328	2,844	463 [2.2e-05]
MW4	University	24,864,872	45,717	17,547 [6.3e-05]
MI1	Commercial	847,884,900	641,378	42,820 [5.1e-04]
PP1	Independent	800,791	6,928	2,321 [1.9e-04]
SU1	ISP	65,733,781	1,029,575	57,942 [1.3e-03]
UI1	University	26,921,525	18,951	519 [1.22e-05]
UI2	University	51,722,823	22,462	1012 [2.12e-05]
UI3	University	46,321,161	22,351	674 [2.57e-05]

Table I summarizes the key characteristics of the NTP data which forms the basis of our study. The NTP servers are

located in 9 different cities, and include a combination of *(1)* 2 different Internet service providers in Chicago (IL), Edison (NJ), and Salt Lake City (UT) resulting in 7 NTP servers, *(2)* 3 commercial NTP servers in Jackson (WI) and Monticello (IA), *(3)* 7 university campus NTP servers in Madison (WI) and Urbana-Champaign (IL), and *(4)* 2 independent/community NTP servers in Atlanta (GA) and Philadelphia (PA). In the table, we observe a wide variation in the number of measurements gathered from each server, as well as a wide range of number of unique clients. Note that all measurements and clients in our study are from IPv4-based networks. In the table, we also include the number of IPv4 prefixes that contain the client population, as well as the fraction of total routable prefixes. The prefix data used to compute the rightmost column in the table comes from CAIDA [13].

B. Address prefix data

Our technique for Internet event detection depends on grouping clients into IP address prefix clusters. The address prefix data that we use is collected as part of CAIDA's prefix-to-AS (Autonomous System) mapping from the RouteViews project [13]. We also use the the IP-to-AS mapping data from Team Cymru [14] to enrich our perspective on widely used prefixes.

C. Datasets for comparative analysis

We use datasets from a number of other efforts to provide perspective on network events that we identify using Tezzeract. As noted above, *validation* is challenging due to the lack of reliable ground truth information. Thus, we draw on two sources in an attempt to understand and contextualize the events detected through our framework. Our goal is to use these comparisons in a targeted fashion to highlight how NTP-based event detection can provide an important and useful and complementary perspective on network events. Specifically, we use *(i)* Internet outage data from ISI's census and survey project [12], [15], and *(ii)* events reported on the outages mailing list [16]. These datasets were all collected contemporaneously with our NTP data and offer a broad perspective about the events that we identify in the NTP logs. Specifically, ISI's census and survey offer a network operation and configuration perspective, whereas events reported on the outages mailing list offer a (limited) operator perspective.

III. Methodology

To identity the events in NTP logs, we developed a framework and implemenation called *Tezzeract* which has two main objectives. The first objective is to identify *all events* in NTP logs, where an *event* is defined as a *significant* change in OWD that affects multiple clients within an IP prefix. Most events such as outages and route changes will manifest in a large increase in OWDs, while other events such as peering updates could manifest in a decrease in OWDs. The second objective is to provide details on characteristics of events in terms of duration, number of clients that experience an event on a per server basis and across servers. To achieve these objectives, Tezzeract consists of two algorithms: *TezzeractClusterGenerator* and *TezzeractEventDetector*, which we describe below.

A. Cluster generator

Tezzeract begins by ingesting OWD data from tightly synchronized clients[1] to generate clusters of NTP clients in a matrix. A *cluster* is simply the largest IP prefix aggregate in which we observe a given NTP client. The *TezzeractClusterGenerator* algorithm takes three inputs: *(a)* NTP logs from tightly synchronized clients, *(b)* IP prefix-to -AS mapping from CAIDA [13], and *(c)* IP address-to-AS mapping from Team Cymru [14].

The algorithm starts by extracting the IP addresses of clients from NTP logs and creates prefix tries using IP prefix/address-to-AS mapping datasets. Next, for every client C that synchronizes time with an NTP server S, the longest matching prefix among the CAIDA and Team Cymru data sources is determined. If a prefix is not already seen, a new cluster is created with the prefix as key and the set of clients for the new cluster is initialized with C. Otherwise, C is added to the set of clients of an existing prefix cluster. The clustering process accomplishes two goals: *(1)* it creates client groupings which naturally relate to Internet routing and management activity, which we hypothesize are commonly related to observed outages and performance disruptions, and *(2)* it reduces the number of dimensions of the matrix on which event detection is applied (see §III-B), thus reducing computational demands.

B. Event identifier

The *TezzeractEventDetector* algorithm generates a matrix of OWD values for every observed prefix cluster. Specifically, for every prefix cluster P, an OWD matrix of dimension t x n is generated, where t is the time bin used to group every client in a row and n is the number of clients in a prefix cluster. From the NTP logs, the polling intervals are extracted and t is determined using the median of minimum polling intervals, which determines the frequency of NTP packet exchanges for individual clients. Subsequently, the start and end epochs are generated from timestamps. Using the epochs, the time dimension of the matrix is determined and an empty matrix is generated.

Next, the algorithm populates the t x n matrix using the OWDs extracted from the NTP packets. Note that our description focuses on a single matrix for brevity. There are actually *two* matrices constructed (and processed in later steps): one for client-to-server (c2s) OWDs and one for server-to-client (s2c) OWDs. For each client C, the corresponding OWD vectors and epoch timestamps are extracted. These timestamps are used to determine the value stored at a particular index in the matrix for a client. If a client has multiple OWDs in a particular time bin, we take the maximum value. In our algorithm, if a particular client has no value for a given time bin due to missing NTP packets, we leave that entry as "NA". We remove all those prefixes with n less than 2 as well as all rows with complete NA values, resulting in a t' x n matrix, where $t' \leq t$.

[1] NTP clients that have tight synchronization with their servers by accurately accounting for and correcting the clock drift. We utilize the technique described in [9] to identify such clients.

The core of our event detection method is based on identifying outliers in the t' x n matrix. We do this by applying Robust Principal Component Analysis (RPCA) [17] with Mahalanobis distance-based thresholding [18]. RPCA is robust to missing data, which is a key aspect of our matrix [19]. This approach allows us to identify events and to characterize them in terms of time duration and number of clients on every individual OWD matrix. We begin by establishing centers of the OWD vectors from the matrix that are projected on an ellipse. Subsequently, the principal components are derived as the eigenvectors of the robust covariance matrix of the projected data points.

RPCA Calibration. Informed by the prior work on the sensitivity of PCA for detecting network anomalies [20], we use cautions in selecting configuration parameters. In particular, the following three parameters are important:

- **Number of principal components** (top_k) that are used for the new projection to reduce dimensions. If we use all n principal components derived from a t' x n matrix, then we are able to account for maximal variance in the data. In this study, we consider the algorithm's sensitivity to choosing the top_k principal components.
- **Scoring distance threshold**, which is also known as the Mahalanobis distance. Similar to prior efforts in this space, we set the outlier detection threshold based on ROBPCA [18], which provides the statistical reasoning for choosing threshold values.
- **Orthogonal distance threshold**, a distance metric that is needed *only* if we choose fewer than n principal components. For k principal components ($k < n$), the value of the threshold is established using ROBPCA [18].

Given n principal components, each of which contributes to some percentage of variance in the NTP data, we select the top_k components with *non-negligible* variance. In our analysis, we considered different variance threshold values for determining how to select the top_k principal components. Specifically, we first select NTP logs from two random days and compute the variance contributed by every principal component for all the clusters across all the servers. Next, we iteratively determine thresholds by ignoring principal components that produce lower variance than the current threshold. The threshold values iterated are 0.5, 1, 2, 5, 10 and 20 (and all thresholds are percentages).

Figure 1 shows the variance threshold along with changes in the number of events detected (bottom) and prefix clusters affected (top) for the MW3 NTP server.[2] From this plot for the MW3 server, we consider a threshold between 5 and 10% to represent a reasonable tradeoff between being too sensitive and treating too many OWD fluctuations as significant events on the one hand, and ignoring what are likely to be important performance disruptions on the other hand. Based on this analysis, we set the top_k variance threshold to be 5% in all the analyses presented in this paper. Finally, the score estimates for determining events are calculated using Mahalanobis distance and Orthogonal distance in the projected space. The outliers are identified by applying thresholds on the scores using the thresholding technique similar to ROBPCA [18].

[2]Other NTP servers exhibited similar variance thresholds and are not shown here due to space constraints.

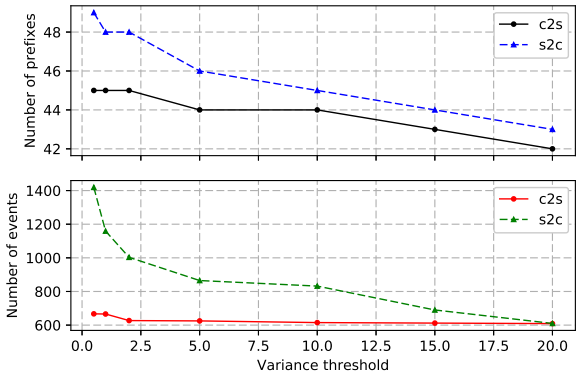


Fig. 1. **Sensitivity analysis of top_k parameter for MW3 server.**

RPCA Implementation. To implement the RPCA component in the event detector, we use the PcaNA R package [21] which consists of many RPCA implementations including Projection Pursuit (PP) [22], Elliptical PCA (EPCA) [10], PCAGRID [23], ROBPCA [18] and Robust Covariance Estimator (RCE) [17]; both ROBPCA and RCE use Minimum Covariance Determinant (MCD). In particular, we use the RCE with MCD as our RPCA for two reasons. First, prior efforts have shown the threshold effectiveness of MCD in ROBPCA on datasets with missing values [24], which is also an issue with our data. Second, in a limited comparison experiment, we evaluated each of the RPCA implementations using a subset of data with validated events and compared the scores. We found that the scores from other methods were either inconsistent or failed to identify the events of interest in comparison with RCE. Furthermore, for the same data, we found that the runtime performance of RCE (0.74s) is over ∼3x faster than the other RPCA methods: EPCA (2.26s), PP (2.83s) and PCAGrid (3.09s). The result is an efficient implementation of *Tezzeract*, where run time on the largest NTP log for one day was 683.79 seconds.

IV. Results

A. Synthetic event tests

To illustrate the efficacy of Tezzeract's event detection capability in a controlled and repeatable fashion, we conduct a series of tests by injecting synthetic events into NTP traces. Our test data is based on an NTP log from a single server (AG1) for a single day (November 15, 2015). We run Tezzeract on the base trace and mark all events detected so they can be removed from further test results. We then modify entries in the trace to simulate events. We consider two factors for injecting an event. First, the duration of the event, and second, the percentage of the clients within a prefix cluster that observe the event (*i.e.,* through expanded OWDs).

Our canonical "event" takes place in a single /24 prefix cluster (256 clients) where at least 20% of the constituent clients exchanged NTP packets with the server. We randomly

increase OWDs on 75% of the observed clients by a factor of 4 to 5 over a period of 7 minutes. Our first test injects an "event" into a /24 prefix cluster in the test trace every 25-minutes. Figure 2-(top) illustrates the OWDs for the synthetic events. Figure 2-(bottom) shows the RPCA scores from Tezzeract, along with the scoring distance threshold. Tezzeract methodology easily identifies all of these injected events, without any false negatives or positives.

Next, we inject a set of random events controlling both the duration and the percentage of clients that experience the event, p. The duration of these events was between 5 minutes to 22 hours. p is varied from 25 to 100%. Similar to the canonical event, we modify existing packets by increasing OWDs and add no new exchanges to the trace. To determine the increase in delay, we consider the inter-packet spacing along with polling intervals to identify the maximum possible delay. Table II shows the results of applying Tezzeract to the modified trace. While no false positives were generated, as the value of p decreases, the number of false negatives climbs—up to 17.9% when only 25% of the clients in the prefix experience the event. Closer examination revealed that all of these events had a duration of less than 20 minutes and thus were not detected simply due to lack of data. We ran many additional experiments with different values for event durations and percentage of clients affected, and the results were consistent with what we report here—no false positives and low false negatives.

TABLE II
Summary of injected events.

Percentage of clients in the prefix	Total number of events injected	Total number of events Tezzeract detected
100	1,000	1,000
75	1,000	983
50	1,000	962
25	1,000	821

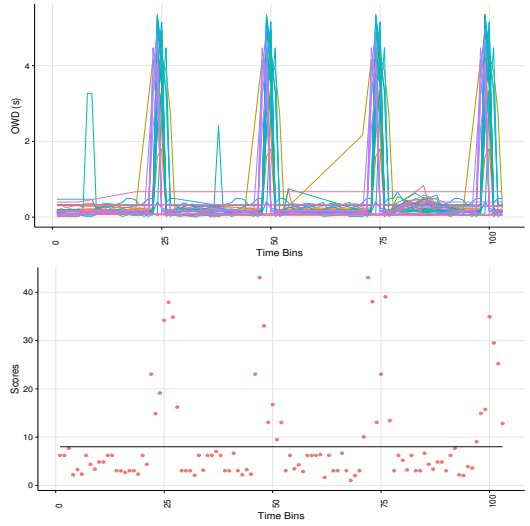


Fig. 2. **OWDs (top) and RPCA scores (bottom) for 7-minute synthetic events with 25-minute inter-event spacing on** `/24` **cluster**

B. Characteristics of events in NTP trace data

We apply Tezzeract to the full set of NTP trace data described in §II.

Individual Servers. Table III shows the summary of the average number of events identified by Tezzeract per day. The number of corresponding prefixes (prefix clusters) that are affected is also shown. The results show that the average number of events identified by Tezzeract varies between NTP servers: from 57k events over 6.7k prefixes (MI1) to 77 events for 8 prefixes (MW1). Comparing the number of prefixes with events (in Table III) with the total number of prefixes reported in Table I, we observe high event occurrences for university-related servers (∼98% for UI3). The results also show that the number of prefixes affected as seen from the ISP servers are fewer than the university servers. We hypothesize the lower number of events observed at the ISP-based NTP servers is due to the narrower reach of client prefixes. Many of these events are likely to be due to daily route changes, which are known to cause temporary increases in packet latencies [25].

TABLE III
Summary of events detected by Tezzeract.

Server ID	c2s		s2c	
	Avg. #events per day	#prefixes affected	Avg. #events per day	#prefixes affected
AG1	4,246	2,342	3,701	2,241
CI1	243	35	216	37
CI2	117	19	118	18
CI3	158	21	129	20
CI4	137	22	146	21
EN1	125	22	128	21
EN2	84	23	92	22
JW1	480	118	701	117
JW2	611	132	944	126
MW1	77	8	75	8
MW2	4,975	1,321	4,868	1,182
MW3	624	80	706	79
MW4	3,586	914	3,276	793
MI1	58,434	6,874	57,396	6,792
PP1	357	151	206	141
SU1	2,520	5,620	1,928	4,954
UI1	2,949	496	2,872	446
UI2	8,570	863	8,128	804
UI3	6,819	664	6,118	627

Figure 3 depicts the number of events and prefixes affected as seen from the PP1 NTP server. The figure shows that the number of events observed for the c2s path of PP1 NTP server is as high as 890; whereas it is 400 events for the s2c path. This highlights well the known issue of Internet routing asymmetries [26] and can serve as evidence for further diagnosis of the events. Interestingly, we observe a drop in the number of events and affected prefixes for the PP1 commercial NTP server over December 25 and 26, 2015. This "Christmas holiday effect" is consistent across all commercial NTP servers, but *not* consistently observed through ISP- and university-based NTP servers. We also find that the number of events detected per day varies based on whether c2s or s2c OWDs are considered once again highlighting the issue of routing asymmetries.

To complement Table III, Figure 4 shows box-and-whiskers plot of event durations for c2s events. Note that we do not show all outliers in this plot to avoid visual clutter. The figure shows that the interquartile ranges for the ISP-based NTP server (*i.e.*, SU1) are very tight. On the contrary, even though the interquartile ranges for certain servers (*e.g.*, MI1 and CI1) are comparable with other NTP servers, their maximum observed event duration exceeds 6000s (∼1.5 hours). The

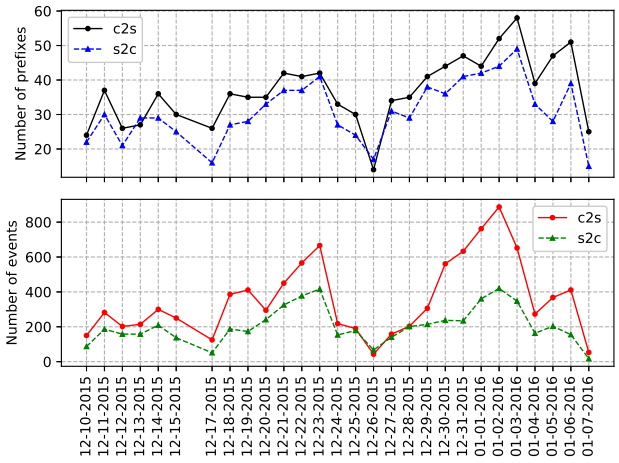


Fig. 3. **Number of events and affected prefixes for PP1**

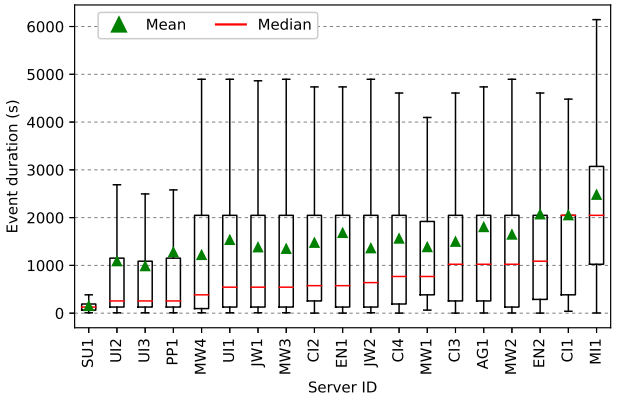


Fig. 4. **Box-and-whiskers plots showing event duration characteristics of c2s events for NTP servers.**

figure also shows that the 75^{th} percentile of event duration for a majority of the NTP servers that we consider is ∼2000s (∼33 minutes). Finally, despite the median of event durations for a majority of servers being very low (*i.e.*, generally less than 15 minutes), we observe that the maximum event durations are fairly high. One explanation for high event duration values is clients with long NTP polling intervals (*e.g.*, 1024s), thus these values are not necessarily reflective of event duration. s2c paths also exhibited similar characteristics. Enhancing event duration estimates is a topic of future work.

Across Servers. Next, we count the unique number of prefixes and events seen across the NTP servers. Figure 5 shows the number of non-overlapping events (y axis) as observed from at least n NTP servers (x axis). All the reported counts are time-aligned *i.e.*, both events and prefixes are simultaneously identified by Tezzeract at one or more NTP servers. We observe over 11.3k and 10.8k unique prefixes from different NTP servers in c2s and s2c directions, respectively. From these prefixes, events from ∼25%, ∼5% and ∼3% of the prefixes are seen across at 2, 3, or 4 NTP servers, respectively. Figure 5 indicates events observed by as many as 10 NTP servers for certain prefixes (*e.g.*, `108.192.0.0/16` belonging to AT&T, Inc.).

Overall, these results demonstrate how NTP can offer a unique Internet-wide perspective. In particular, Figure 5 shows how prefixes could be simultaneously monitored from multiple servers. Moreover, when events are observed from multiple servers for any given prefix at the same time, they can provide a natural self-consistency check and aid in diagnosis.

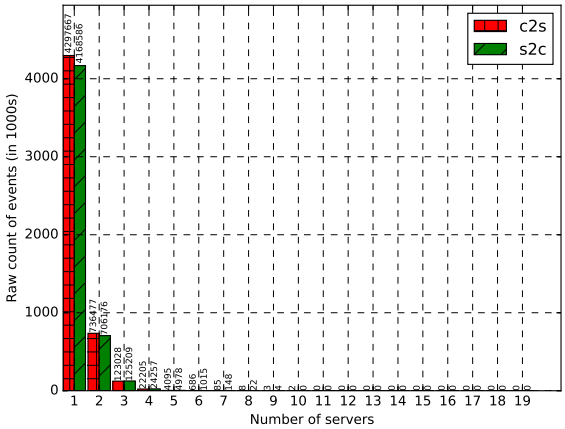


Fig. 5. **Number of events seen across multiple servers.**

C. Consistency checking with active probe-based data

Next, we compare and contrast the events identified by Tezzeract versus those identified by the ongoing Internet-wide Census and Survey project at ISI [12]. The ISI Census and Survey project is an example of what we consider a *core-to-edge active probing system*. Tezzeract, on the other hand, has a specific goal of identifying events based on passive observation of NTP traffic and which provides *both* core-to-edge and edge-to-core perspectives. Given the differences between the two methods and given the paucity of ground truth information to validate the identified events, the comparison of the number of events and affected prefixes identified by Tezzeract vs. census and survey should be interpreted as a way to strengthen and learn from each other. That is, Tezzeract can be used to provide a better perspective on edge-to-core events as the majority of clients behind network address translators are oblivious to active probing efforts.

Our comparison algorithm for the ISI census and survey data takes two inputs: *(1)* the ISI-identified events (*i.e.*, those recorded in /24 prefixes through active probing) and *(2)* the events identified using Tezzeract. Note that both inputs also contain the timeline of event occurrences. Given these inputs, the algorithm populates a trie with Tezzeract-identified event prefixes and their corresponding timelines (*i.e.*, event start and end times) since the events identified by Tezzeract contain prefixes larger than /24. Next, for each ISI event reported, we check whether the prefix associated with the ISI event is in the trie (or is contained within a prefix in the trie). If a match is found, the algorithm finds the best matching NTP event corresponding to an ISI event by comparing the start time of each NTP event with the ISI event. At the end of this process, we obtain a one-to-one NTP and ISI event *match list*.

Given the match list with two start and end times, say <I1, I2> from ISI and <N1, N2> from Tezzeract, the algorithm assigns the events into one of 11 categories based on the conditions given in Table IV. For categories 1 to 9, we have

some form of overlap between Tezzeract- and ISI-identified events, whereas for categories 10 and 11, there are no overlaps. For categories 10 and 11, we use a pre- and post-match window of 1 hour.

TABLE IV
Categories used for comparing Tezzeract- vs. ISI-identified events.

Category	Condition
1	I1 = N1 and I2 = N2
2	I1 = N1 and I2 < N2
3	I1 = N1 and I2 > N2
4	I1 < N1 and I2 = N2
5	I1 < N1 and I2 > N2
6	I1 < N1 and I2 < N2
7	I1 > N1 and I2 = N2
8	I1 > N1 and I2 > N2
9	I1 > N1 and I2 < N2
10	I1 > N2
11	I2 < N1

TABLE V
Summary of Tezzeract events comparison with ISI events.

Server ID	OWD direction	Total # of ISI event matches with Tezzeract	Total exact /24 matches	Total C1 to C9	C10 and C11
AG1	c2s	68,895	123	48,198	20,697
	s2c	62,460	113	45,228	17,232
CI1	c2s	3,408	2	2,785	623
	s2c	2,773	1	2,317	456
CI2	c2s	1,569	0	1,185	384
	s2c	1,460	6	1,104	356
CI3	c2s	3,513	0	2,958	555
	s2c	2,409	1	2,088	321
CI4	c2s	2,132	6	1,645	487
	s2c	2,306	6	1,890	416
EN1	c2s	1,188	8	980	208
	s2c	1,198	8	994	204
EN2	c2s	807	5	619	188
	s2c	1,077	6	874	203
JW1	c2s	6,988	13	6,185	803
	s2c	8,555	16	7,487	1,068
JW2	c2s	6,657	0	5,533	1,124
	s2c	9,736	0	7,826	1,910
MW1	c2s	1,065	0	756	309
	s2c	1,053	0	725	328
MW2	c2s	79,330	290	55,548	23,782
	s2c	73,749	322	52,020	21,729
MW3	c2s	8,715	63	6,278	2,437
	s2c	10,537	69	7,406	3,131
MW4	c2s	55,406	118	38,038	17,368
	s2c	47,793	131	33,854	13,939
MI1	c2s	959,455	2,667	786,951	172,504
	s2c	951,396	2,589	784,808	166,588
PP1	c2s	3,628	3	2,733	895
	s2c	2,875	4	2,307	568
SU1	c2s	17,717	71	7,606	10,111
	s2c	14,194	47	6,346	7,848
UI1	c2s	32,477	142	23,133	9,344
	s2c	30,849	143	22,362	8,487
UI2	c2s	76,328	261	49,487	26,841
	s2c	72,151	279	48,181	23,970
UI3	c2s	59,919	151	39,339	20,580
	s2c	54,913	143	36,638	18,275

Table V shows the comparison of Tezzeract-identified events (for both c2s and s2c directions) versus events identified by census and survey. The table highlights the number of ISI events which we were able to match with Tezzeract events, the number of exact /24 matches (cases where Tezzeract's prefix cluster is a /24, thereby enabling a direct one-on-one comparison with the ISI outage event), and the number of events under various categories. From this table, we see that as much as ∼67% of the events identified by Tezzeract from the CI2 NTP server match with those identified by census and survey. We further observe that on average, over 66% of the Tezzeract-identified events overlap in terms of event timelines across all the servers (in C1 to C9 category); the remaining (∼33%) of Tezzeract-identified events occur within a match window of one hour, either *before* or *after*, those identified by ISI (see C10 and C11). Next, we see that there are more /24 exact matches between the Tezzeract and ISI for commercial- and university-based NTP servers in comparison with the ISP counterparts. This may simply reflect the fact that NTP clients that contact the ISP servers typically come from larger routing prefix aggregates. Lastly, we observe that Tezzeract finds ∼63% new events, on average, across all the NTP servers considered in this study. This is likely due to the reach of NTP clients vs. the probing cycle of the ISI servers.

The unique events that were identified by Tezzeract consisted of both network prefixes covered as part of the ISI survey, as well as prefixes which were not reachable via standard active probing methods. The median duration of these events is approximately 20 minutes, which is similar to the overall median duration. Tezzeract was able to identify many long duration events, observed by multiple clients within the affected prefix clusters. The top two longest duration events observed lasted for approximately 2 and half hours and affected the following prefixes: `204.93.0.0/19` which belongs to AS 698 (University of Illinois) and `54.186.0.0/15` which belongs to AS 16509 (Amazon.com, Inc.). These events were observed by 53 and 74 unique clients belonging to the corresponding prefix clusters. These events are likely to be outages similar to those discussed below.

D. Consistency checking with other sources

Finally, we compare events identified by Tezzeract with public reports. In particular, we rely on the Outages mailing list [16] to further enhance the confidence in the events identified by Tezzeract. To illustrate with an example, we consider the outage event [27] discussed by Level3's (now CenturyLink) administrators in the Outages mailing list as ground truth. On December 15, 2015 an outage event occurred because of a router addition that impacted multiple users and businesses between Chicago, IL and Atlanta, GA.

To evaluate whether the events identified by Tezzeract coincide with this known outage, we first selected all the events identified on December 15, 2015, resulting in 148,322 events. Similar to our comparison with ISI event data, we use four timestamps: *(1)* two from each event (start and end) identified by Tezzeract, and *(2)* start and end timestamps derived from the Outages mailing list [27], which are December 15, 2015 18:18 GMT and December 15, 2015 19:20 GMT. Next, we find the time-alignment category under which the Tezzeract-identified events fall with respect to [27] (see Table IV). If the events fall under C1 to C9, we note those events; otherwise, we ignore those events. For the events noted, we extract the list of client IP addresses and their corresponding geographic coordinates using MaxMind's IP Geolocation service [28]. Subsequently, we match the geographic locations from [16] and the ones obtained above.

From the 148k events initially identified, the process outlined above results in 1,104 events overlap with the ground

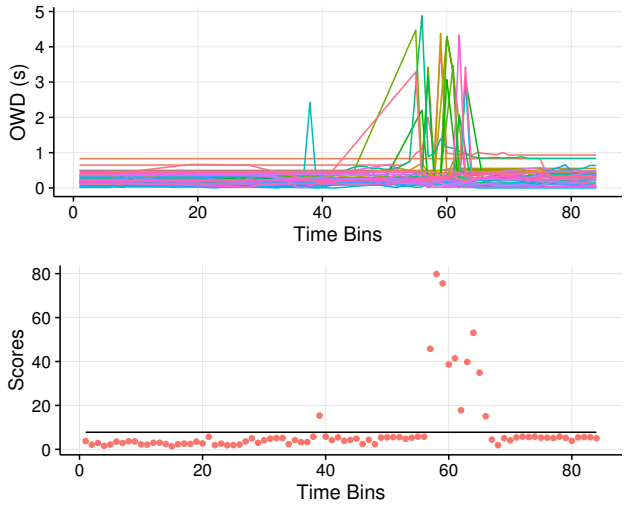


Fig. 6. **Level3 outage event identified by Tezzeract affecting AS 32748.**

truth [27] in terms of event timelines, prefixes and geographic locations. Figure 6 shows the OWD spikes (top) and scores for clients (bottom) in one of the prefixes affected. The OWD spikes represented in Figure 6-(top) contain 43 unique clients from the prefix `208.117.0.0/18` which belongs to AS 32748 (Steadfast.net), a peer of Level3. The results depicted in this plot highlight the effectiveness of Tezzeract in identifying Internet outages. In particular, we observe that the effect on NTP-derived OWDs is extreme for clients within each of these ASes, underscoring the severity of the event. Tezzeract can also offer an alternate perspective to BGP-based event detection tools [29][3].

V. Related Work

Prior research on network outages has considered various perspectives including rerouting or routing anomalies caused by failures in *core and transit networks*, connectivity outages and performance impairments for customers in the *network edge*, and *service outages*.

Core and transit network outages. Core network failures and performance anomalies have been examined using both passively collected data sources and active measurement. Analysis of inter-domain (BGP) and intra-domain (*e.g.*, IS-IS, OSPF, etc.) routing updates have formed the basis of many of the studies based on passive data collection, *e.g.*, [30]–[32]. In a related vein, Banerjee *et al.* used the outages mailing list [16] as the basis for evaluating core network failures [33]. Yet another source of passive data for detecting and analyzing wide-area faults has been through analysis of background radiation traffic [34]. Active measurement techniques have also been widely used to detect routing loops and other anomalies and path failures [35]. Tomographic techniques have also been developed to actively probe a network in order to detect faults and localize them to particular links or subpaths [36].

Outages at the network edge. Active measurement has been the dominant technique for detecting failures at the edge of the network. Periodic pings combined with analysis of BGP updates were used to trigger traceroutes to verify and monitor edge network outages in Hubble [37]. The study and ongoing data collection by Quan *et al.* employ low-rate pings to the entire IPv4 address space, and detect outages and disruption events through a Bayesian formulation [4]. In our study, we use a subset of these data for comparison and validation. In their study, Padmanabhan *et al.* study the response time to pings across the IPv4 Internet and find that 5% of responses from 5% of addresses take at least 5 seconds to arrive [38]. This finding has important implications for the design of any active measurement-based system that uses the lack of responses to identify outages. Our event detection strategy is based on *passively* collected NTP data.

PCA-based analysis of network data. Identifying events of interest from streams of Internet data using PCA has been of interest to the measurement community for many years. This includes applying PCA to detect BGP anomalies [39], network traffic anomaly detection and diagnosis [2], [40], network monitoring and anomaly detection [41], and network diagnosis [42]. Issues with PCA-based methods are pointed out by [43], [44]. Similarly, sensitivity of PCA to calibration and its corresponding implications to anomaly detection are discussed by Ringberg *et al.* [20]. Our event detection approach applies Robust PCA, instead of PCA, on NTP traces and demonstrates a new, unique perspective for Internet-level event detection without additional infrastructure.

VI. Summary and Future Work

Understanding the scope and nature of unexpected network events is important for effective management and operation of communication networks. In this paper, we address the problem of Internet event detection by developing a novel framework and implementation called Tezzeract. Our framework uses one way delays (OWDs) extracted from Network Time Protocol (NTP) packet exchanges and therefore does not require any new infrastructure to be deployed, or any additional network traffic and offers the opportunity for a broad perspective on events. Our algorithm for identifying events is based on Robust PCA, which is resilient to noisy data that is typical in the Internet measurements including NTP data. Our implementation of Tezzeract produces a characterization of events including the number of NTP clients affected, event duration, and prefix(es) affected.

We assess Tezzeract in a series of controlled experiments based on injecting synthetic events in an NTP trace. We find that Tezzeract is highly accurate in reporting with a false negative rate related to measurement reach. We then apply Tezzeract to a large NTP data set collected from 19 servers in the US. We find that the average number of events per day varies widely across the set of servers, as do the event durations. We observe that the median event duration for most servers is approximately 20 minutes or less, but that the distribution is skewed, with quite a few very long events (*e.g.*, more than 1 hour). We also find that a considerable number of events are observed at more than 1 NTP server, *e.g.*, 25% of events are observed at 2 servers, and that we observe

[3]These results are not shown here due to space constraints.

some events at 10 of the 19 servers indicating significant impact across clients. We compare the events detected through Tezzeract with event data collected through the ongoing ISI census and survey project and find that between 21–67% of events that are detected by Tezzeract are also identified by the ISI system, but that Tezzeract also identifies new events. We also evaluate events detected by Tezzeract in relation to a reported outage. We observe that the event is identified through different sets of NTP clients on different prefixes, and but that the different client populations are impacted in similar ways.

In ongoing work, we are examining additional ways to corroborate and gain perspective on the events detected by Tezzeract, and are exploring ways to automate this process. We are also examining events recognized by *decreases* in OWDs, which is a simple extension of Tezzeract. Finally, we are also examining the possibility of performing real-time event detection and analysis.

ACKNOWLEDGMENTS

We thank the NTP operators for providing the datasets. This work is supported by NSF grants CNS-1703592, DHS BAA 11-01, AFRL FA8750-12-2-0328. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, DHS, AFRL or the U.S. Government.

REFERENCES

[1] "Effects of Catastrophic Events on Transportation System Management and Operations: Howard Street Tunnel Fire Baltimore City, Maryland," https://ntl.bts.gov/lib/jpodocs/repts_te/13754.html.

[2] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," *ACM SIGCOMM*, 2005.

[3] P. Barford, N. Duffield, A. Ron, and J. Sommers, "Network performance anomaly detection and localization," in *IEEE INFOCOM*, 2009.

[4] L. Quan, J. Heidemann, and Y. Pradkin, "Trinocular: Understanding Internet reliability through adaptive probing," in *ACM SIGCOMM Computer Communication Review*, 2013.

[5] B. Eriksson, R. Durairajan, and P. Barford, "Riskroute: A framework for mitigating network outage threats," in *ACM CoNEXT*, 2013.

[6] M. Zhang, C. Zhang, V. S. Pai, L. L. Peterson, and R. Y. Wang, "Planetseer: Internet path failure monitoring and characterization in wide-area services." in *OSDI*, 2004.

[7] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan, "Latlong: Diagnosing wide-area latency changes for cdns," *IEEE Transactions on Network and Service Management*, 2012.

[8] R. Durairajan and S. Mani and J. Sommers and P. Barford, "Time's Forgotten: Using NTP to Understand Internet Latency," in *ACM HotNets*, 2015.

[9] R. Durairajan, S. K. Mani, P. Barford, R. Nowak, and J. Sommers, "TimeWeaver: Opportunistic One Way Delay Measurement via NTP," https://arxiv.org/abs/1801.02123, 2018.

[10] N. Locantore, J. Marron, D. Simpson, N. Tripoli, J. Zhang, K. Cohen, G. Boente, R. Fraiman, B. Brumback, C. Croux *et al.*, "Robust Principal Component Analysis for Functional Data," *Test*, 1999.

[11] M. Hubert, P. Rousseeuw, and T. Verdonck, "Robust pca for skewed data and its outlier map," *Computational Statistics & Data Analysis*, 2009.

[12] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister, "Census and Survey of the Visible Internet (extended)," *ISI-TR-2008-649*, 2008.

[13] "CAIDA Routeviews Prefix to AS mappings Dataset (pfx2as) for IPv4 and IPv6." https://www.caida.org/data/routing/routeviews-prefix2as.xml.

[14] "Team CYMRU's IP to ASN mapping." http://www.team-cymru.org/IP-ASN-mapping.html.

[15] "USC/LANDER Project. Internet Outage Dataset, PREDICT ID: USC-LANDER/internet_outage_adaptive_a22all-20151001 and USC-LANDER/internet_outage_adaptive_a23all-20161001," http://www.isi.edu/ant/lander.

[16] "Outages Mailing List." https://puck.nether.net/mailman/listinfo/outages.

[17] C. Croux and G. Haesbroeck, "Principal Component Analysis based on Robust Estimators of the Covariance or Correlation matrix: Influence Functions and Efficiencies," *Biometrika*, 2000.

[18] M. Hubert and P. J. Rousseeuw and K. Vanden Branden, "ROBPCA: A New Approach to Robust Principal Component Analysis," *Technometrics*, 2005.

[19] S. Serneels and T. Verdonck, "Principal Component Analysis for Data Containing Outliers and Missing Elements," *Computational Statistics and Data Analysis*, 2008.

[20] H. Ringberg and A. Soule and J. Rexford and C. Diot, "Sensitivity of PCA for Traffic Anomaly Detection," *SIGMETRICS*, 2007.

[21] "R package: Classical Or Robust Principal Components For Incomplete Data." https://www.rdocumentation.org/packages/rrcovNA/versions/0.4-8/topics/PcaNA.

[22] C. Croux and A. Ruiz-Gazen, "High Breakdown Estimators for Principal Components: The Projection-Pursuit Approach Revisited," *Journal of Multivariate Analysis*, 2005.

[23] C. Croux, P. Filzmoser, and M. R. Oliveira, "Algorithms for Projection–Pursuit Robust Principal Component Analysis," *Chemometrics and Intelligent Laboratory Systems*, 2007.

[24] C. Pascoal, M. R. De Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, "Robust feature selection and robust pca for internet traffic anomaly detection," in *IEEE INFOCOM*, 2012.

[25] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," in *ACM SIGMETRICS Performance Evaluation Review*. ACM, 2007.

[26] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A Measurement Study of Internet Delay Asymmetry," in *PAM*, 2008.

[27] "[outages] Level3 Chicago?" https://puck.nether.net/pipermail/outages/2015-December/008572.html.

[28] "MaxMind IP Geolocation Service." https://www.maxmind.com/.

[29] M. Syamkumar, R. Durairajan, and P. Barford, "Bigfoot: A Geo-based Visualization Methodology for Detecting BGP Threats," in *IEEE Symposium on Visualization for Cyber Security*, 2016.

[30] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, 1998.

[31] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental study of Internet stability and backbone failures," in *International Symposium on Fault-Tolerant Computing*, 1999.

[32] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Transactions on Networking*, 2008.

[33] R. Banerjee, A. Razaghpanah, L. Chiang, A. Mishra, V. Sekar, Y. Choi, and P. Gill, "Internet outages, the eyewitness accounts: Analysis of the outages mailing list," in *PAM*, 2015.

[34] K. Benson, A. Dainotti, K. C. Claffy, and E. Aben, "Gaining insight into AS-level outages through analysis of Internet background radiation," in *IEEE INFOCOM workshops*, 2013.

[35] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM transactions on Networking*, 1997.

[36] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in *ACM CoNEXT conference*, 2007.

[37] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. E. Anderson, "Studying Black Holes in the Internet with Hubble," in *NSDI*, 2008.

[38] R. Padmanabhan, P. Owen, A. Schulman, and N. Spring, "Timeouts: Beware surprisingly high delay," in *ACM IMC*, 2015.

[39] K. Xu and J. Chandrashekar and Z.L. Zhang, "A First Step toward Understanding Inter-domain Routing Dynamics," in *ACM SIGCOMM workshop on Mining network data*, 2005.

[40] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-wide Traffic Anomalies," in *ACM SIGCOMM*, 2004.

[41] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, and G. Maciá-Fernández, "PCA-based Multivariate Statistical Network Monitoring for Anomaly Detection," *Computers & Security*, 2016.

[42] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone, "MIND: A Distributed Multi-Dimensional Indexing System for Network Diagnosis," in *IEEE INFOCOM*, 2006.

[43] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural computation*, 1998.

[44] M.E. Tipping and C.M. Bishop, "Mixtures of Probabilistic Principal Component Analyzers," *Neural computation*, 1999.

State Acquisition in Computer Networks

Ruairí de Fréin
School of Electrical and Electronic Engineering
Dublin Institute of Technology, Ireland

***Abstract*—We establish that State Acquisition should be performed in networks at a rate which is consistent with the rate-of-change of the element or service being observed. We demonstrate that many existing monitoring and service-level prediction tools do not acquire network state in an appropriate manner. To address this challenge: (1) we define the rate-of-change of different applications; (2) we use methods for analysis of unevenly spaced time series, specifically, time series arising from video and voice applications, to estimate the rate-of-change of these services; and finally, (3) we demonstrate how to acquire network state accurately for a number of real-world traces using Greedy Acquisition. The accuracy of State Acquisition is improved when it is performed at a rate which is consistent with its rate-of-change. An improvement in representation accuracy of one order of magnitude is achieved for voice and video streaming applications; this improvement does not incur any additional bandwidth or storage cost.**

***Index Terms*—State Acquisition; Network monitoring; Spectral analysis; Period detection.**

I. Introduction

MANY network management tools use Linux tools for State Acquisition to acquire inputs for higher-level functions such as network monitoring algorithms [1], service level prediction algorithms [2], [3] and resource allocation routines [4]. Some examples of these acquisition tools include the System Activity Report (SAR) [5], Nagios [6] and top [7] (in association with tools such as netstat and dropwatch). Monitoring tools that use these acquisition methods promise to deliver notifications to the user if the aggregate of some metric is above a threshold across an entire infrastructure or on a per machine basis [6]. With respect to a cluster of machines, aggregates in terms of the average, maximum, minimum or sum are computed [1] as a function of time and/or across machines or instances [8]. These types of notifications are given for a number of different metrics, for example, revenue or data center temperature. Statistics are then absorbed by tools such as StatsD [9] and representative charts are generated every 10s for example (using tools such as Graphite). This paper considers the validity of current State Acquisition approaches, which acquire the data for higher-level functions such as monitoring, learning, graphing and problem diagnosis. In particular, we examine methods for acquiring and aggregating network metrics in voice and video applications [10].

To fix ideas, State Acquisition is defined as a function that measures the state of a network or service and converts this state into a numeric value; its role is different to monitoring, which is a means for making the acquired state available to the network manager. Fig. 1 demonstrates a generic acquisition-monitoring set-up, which is representative of many scenarios. An acquisition function observes an element's state every T_a seconds; a monitoring agent submits a report to the network manager every T_m seconds based on the acquisition agent's observations. In many applications, the acquisition and monitoring functions use the same time-step, $T_a = T_m$; the acquisition agent and monitoring agent are one and the same.

The design of network monitoring algorithms (such as [1]) does not focus on the role of metric acquisition. Linux comes with widely used metric acquisition routines. The assumption is that existing tools can be relied upon to acquire metrics; the role of the monitoring protocol is to determine when to aggregate the metrics and to deliver them to the network manager. The confidence the monitoring and learning communities have in these acquisition routines is unwarranted – these routines may not be sufficiently accurate for modern network monitoring and learning protocols [11]. The performance of a monitoring function depends on the quality of the data that it consumes. The increase in dynamicity and heterogeneity of modern networks [12], which makes networks harder to manage, exacerbates this problem.

The problem with off-the-shelf metric acquisition routines is that many of them are periodic with a default resolution of 1 second, for example SAR [5]. Consequently, many network monitoring routines monitor the network with this temporal precision limitation [13]. This was one of the reasons (scalability issues contributed also) why SoundCloud developers developed their own monitoring solution, Prometheus [8]. Similar in spirit to our approach, Prometheus stores all data that is periodically pulled from SoundCloud's [14] instrumented microservices architecture as time series, where time-stamps have a millisecond resolution and values are 64-bit floats. In order to save disk space, Nagios XI stores performance data in a Round Robin Database, which consists of performance metrics periodically averaged over $1, 5, 30$ and 360 minute time-steps. We examine if these periodic acquisition approaches yield sufficient accuracy.

In the case of event-based monitoring and learning routines using SAR [5], there may be a lag of up to 1 second between the occurrence of an event and the time a monitoring report is sent in response to it [3]. The descriptor, "event-based", is inappropriate. The authors of [2] and [3] use SAR to acquire kernel metrics from a video server once per second in order to perform client service level prediction, dealing with load-

ISBN 978-3-903176-08-9

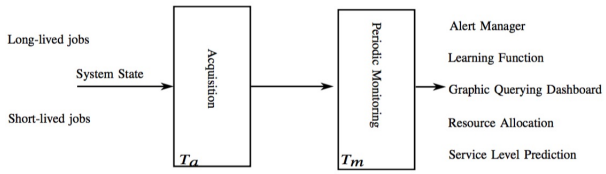


Fig. 1. The role of state acquisition in the monitoring ecosystem: metrics are pushed every T_a seconds to a periodic monitoring gateway, with period T_m, from instrumented systems delivering either long or short lived jobs. Periodically acquired observations are stored locally, rules are applied to them to generate new time series; alerts are triggered; they contribute to monitoring or querying dashboards; or they are supplied to learning APIs. The performance of these higher-order functions depends on quality of the acquired data.

effects in particular in [15]. For a video and voice resource allocation application the authors of [4] acquired measurements of resource parameters (CPU, RAM, latency and call drops) every 15s from a Clearwater cloud ISM test-bed, using SNMP and Cacti [16] to determine how physical resources could be dynamically allocated to virtualized network functions. More generally, RTCP [17] is commonly used to provide out-of-band statistics for RTP sessions by periodically reporting on packet counts, packet loss, packet delay variation, and round-trip delay time to participants in a streaming multimedia session [18]. The recommended minimum RTCP report interval per station is 5 seconds. Nagios' Remote Plugin Executor periodically polls the agent on the remote system for disk usage and system-load statistics. Now that we have established the central role periodic acquisition and monitoring plays in networks, we examine the efficacy of periodic State Acquisition.

We query the acquisition resolution required to monitor the quality of service received by the client in a video and in a voice session. Minimizing network bandwidth usage plays a role in determining the acquisition periods of $T_a = 1, 5$ and 15 seconds used in the approaches above. Consuming bandwidth with monitoring reports is undesirable; however, reporting the system state inaccurately is perhaps even more undesirable than not reporting at all. We examine this trade-off between accuracy and bandwidth usage, but also consider whether crucial characteristics of each trace have been preserved by State Acquisition at different points of this trade-off.

This paper is organized as follows. In Section II, we provide a framework for describing state acquisition methods. In Section III, we consider periodic acquisition and the effects of performing faster acquisition empirically and motivate Greedy Acquisition. In Section IV, we discuss rate of change estimation. In Section V, we describe a Greedy Acquisition algorithm which performs acquisition in a manner which is consistent with the rate of change of the observed process. In Section VI, we perform a numerical evaluation of current acquisition methods, and the Greedy Acquisition method.

II. State Acquisition: Unevenly spaced samples

The time series observed in networks are unevenly spaced. They consist of a sequence of observation time and value pairs (t_n, x_n) with strictly increasing observation times. When audio or video frames (that are streamed from a SoundCloud or Youtube server) are observed at a client machine, the time-stamps of frame arrivals and the frame-sizes, (t_n, x_n), can be useful for service outage detection. Fig. 2 illustrates a frame-arrival time-series at a client machine when a podcast is streamed. When the network is sufficiently well provisioned, this process has a periodic component. The server periodically sends the client a segment of data, which the client then plays-out to the user at the sampling rate of the original recording [19]. Important statistics such as jitter and packet loss rates may be computed from this time-series [18]; however, network anomaly diagnosis may be performed, or service-level prediction [3], if these time-series are gathered from multiple network elements at one monitoring/learning gateway [10]. This motivates the question: do the acquisition methods that currently operate at clients allow us to determine (1) the rate of change of the observed process and (2) the period of this process illustrated in Fig. 2? Knowledge of these parameters is crucial for building a learning function that can predict future network behaviour (cf. [3]).

We define a framework for analyzing unevenly spaced time series to answer this question. For $N \geq 1$, the space of strictly increasing time sequences of length N is denoted: $\mathbb{T}_N = \{(t_1 < t_2 < \ldots < t_N) : t_n \in \mathbb{R}, 1 \leq n \leq N\}$. More generally, the space of strictly increasing time sequences is denoted, $\mathbb{T} = \cup_{N=1}^{\infty} \mathbb{T}_N$. The observation values, x, in computer networks are real-valued, $\mathbb{R}^N$. Bringing these ideas together, the space of real-valued, unevenly spaced time series of length N, is $\mathcal{T}_N = \mathbb{T}_N \times \mathbb{R}^N$. Finally, the space of real-valued unevenly spaced time series is $\mathcal{T} = \cup_{N=1}^{\infty} \mathcal{T}_N$. We often need to quantify the number of observations in some sequence x, e.g. $N = |x|$. The sequence of observation times is denoted, $T(x) = (t_1, \ldots, t_N)$, and the sequence of observation values of x is $V(x) = (x_1, \ldots, x_N)$.

Time series acquisition methods (used by [5],[6] and [7]) are used to summarize the performance of network entities, so that at times $t = nT_a$, performance can be quantified. Here n is an integer that denotes the acquisition time index. There are a number of different acquisition methods. The extracted metrics are passed to a monitoring protocol. Many methods do not yield data which is of sufficient quality. In this paper we address the problem of how to extract a suitable metric from unevenly spaced system events. In short, if we observe some process at times $T(x)$, we investigate what value we should use to represent this process at time $t \notin T(x)$, which is typically not an observation time.

Observation methods: Firstly, we take the previous observation as our estimate. Secondly, we take the next observation as our estimate; and thirdly, we interpolate between the two. For a time series $x \in \mathcal{T}$ and a point in time $t \in \mathbb{R}$, typically not an observation time, the most recent observation time is

$$p(t) =: \begin{cases} \max(s : s < t, s \in T(x)), & \text{if } t \geq \min(T(x)) \\ \min(T(x)), & \text{otherwise.} \end{cases} \quad (1)$$

The next available observation time is

$$n(t) =: \begin{cases} \min(s : s \geq t, s \in T(x), & \text{if } t \leq \max(T(x)), \\ +\infty, & \text{otherwise.} \end{cases} \quad (2)$$

For $x \in \mathcal{T}$ and $t \in \mathbb{R}$, $x(p(t))$ is the previous observation value of x at time t, $x(n(t))$ is the next observation value of x at t, and $x_l(t) = (1 - \omega(t))x(p(t)) + \omega(t)x(n(t))$ where

$$\omega(t) = \begin{cases} \frac{t-p(t)}{n(t)-p(t)}, & \text{if } 0 < n(t) - p(t) < \infty \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

is the linearly interpolated value of x at time t. These acquisition schemes are called last-observation, next-observation and linear interpolation, respectively. We adopt the convention that $x(t) = x(n(t)) = x_l(t)$ when $t \in T(x)$. The interpolated signal $x_l(t)$ is a continuous piece-wise-linear function.

Local-in-time statistics: It is convenient to consider **short-time** observations of these time series in order to generate local-in-time statistics of network behaviour. The time series generated in networks over a closed interval, which starts at time s and ends at time t, $[s, t]$, where $s < t$ is

$$x\{s, t\} = ((t_n, x_n) : s \leq t_n < t, 1 \leq n \leq N). \quad (4)$$

One acquisition approach (cf. [1]) is to apply the max operator to the values in a closed interval $[s, t]$ and to slide this interval, by some step-size, over the entire signal. The maximum signal value in the closed interval $[s, t]$ is denoted

$$a_s(t) = \max V(x\{s, t\}) \quad (5)$$

The minimum value in this interval

$$m_s(t) = \min V(x\{s, t\}) \quad (6)$$

The average value in a closed interval is commonly used to summarize system performance [5],[6] and [7].

$$\mu_s(t) = \frac{1}{|X\{s, t\}|} \sum_{n=n(s)}^{p(t)} x_n \quad (7)$$

The maximum, minimum and average statistics are reported by [20]. Other higher-order statistics are computed in a similar manner, and may be of use to network monitoring applications. The time duration between consecutive observations of x is useful for determining the rate of change of the time series

$$\Delta t(x) = ((t_{n+1}, t_{n+1} - t_n) : 1 \leq n \leq N - 1). \quad (8)$$

III. Acquire Faster: Periodic Acquisition

How representative are $x(p(t)), x_l(t)$ and $\mu_x(t)$ of the data they summarize? The acquisition methods used in [1], [2], [3] and [4], are periodic, and use one of the approaches above, at a rate of 1Hz or greater to quantify network behaviour. Given the increased dynamicity and complexity of modern networks this warrants further inspection. Is a periodic approach good enough? Is an acquisition rate of 1Hz appropriate? To determine the rate of change of unevenly spaced signals, we examine their spectral content by computing a Power Spectral Density (PSD) estimate of x.

The Lomb-Scargle method [21] generates a PSD estimate of unevenly spaced time series, $P : x_n \in \mathbb{R} \mapsto \hat{x}(\omega) \in \mathbb{R}_+$, without the need to invent otherwise non-existent, evenly spaced data. The underpinning assumption is that the signals are periodic. The approximate periodicity of unevenly spaced time series assumption is realistic when we consider the rate of arrival of audio or video frames during a streaming session (Fig. 2).

To reduce notation we subtract the mean from the signal x, and then determine the normalized spectral content of $x \in \mathcal{T}$

$$\hat{x}(\omega) = P(x)\{\omega\} = \frac{1}{2\sigma^2} \times \left\{ \frac{\left[\sum_n x_n \cos(\omega(t_n - \tau))\right]^2}{\sum_n \cos^2(\omega(t_n - \tau))} + \frac{\left[\sum_n x_n \sin(\omega(t_n - \tau))\right]^2}{\sum_n \sin^2(\omega(t_n - \tau))} \right\} \quad (9)$$

at the frequencies ω, where $\sigma^2 = \frac{1}{N-1} \sum_{n=1}^{N} (x_n - \mu)^2$. The following time offset is used to guarantee the time in-variance of the computed spectrum

$$\tan(2\omega\tau) = \frac{\sum_{n=1}^{N} \sin(2\omega t_n)}{\sum_{n=1}^{N} \cos(2\omega t_n)}. \quad (10)$$

We assume that network-generated time-series are base-band or low-pass signals. Empirical evidence in Section VI supports this assumption. To fix ideas, we stream a podcast from SoundCloud [14]. The average received bit-rate of the podcast is 104, 991kbps. We capture an unevenly spaced time series which consists of the time-stamps and sizes (in bits) of each frame, (t_n, x_n) respectively, in the stream using TCPDUMP [22]. We filter out the time-stamp and frame sizes and plot this unevenly spaced time series in Fig. 2. This trace has a periodic component – every approximately 10 seconds, a number of 12kbits frames are delivered.

In Fig. 3 we plot the PSD estimate of this unevenly spaced time series. The component with the highest PSD is $\approx$ .7Hz. By inspection of Fig. 2, we determine that the fundamental frequency of this trace is < 1Hz. The upper envelope of the PSD falls to 0dB/Hz at $\approx$ 100Hz. There are other higher-frequency components in Fig. 2. Consequently, in Fig. 3 we illustrate where the PSD of the trace has fallen by 50dB from its peak value – this occurs at $\approx$ 108 Hz – to demonstrate a range of frequencies of interest to monitoring applications.

Do current time series acquisition approaches preserve this important information about the rate of change and the period of frame delivery? To answer this question, we acquire state values every T_a = 1s, at times $t \in \mathbb{N}$ and examine the PSD of the resulting traces. The time period T_a = 1s is representative of the period used in [1], [2], [3] and [4]. The set of non-negative integers is $\mathbb{N}$. The underlying process we want to acquire an accurate representation for, is acquired by taking (1) the last sample closest to some sampling time, $x(p(t)) \mid t \in \mathbb{N}$; (2) the average value over the last sampling period, $\mu_{t-1}(t) \mid t \in \mathbb{N} \setminus 0$, where t is in the set of positive integers; and finally, (3) the linear interpolated value $x_l(t) \mid t \in \mathbb{N}$. We use a stem to denote the position and height

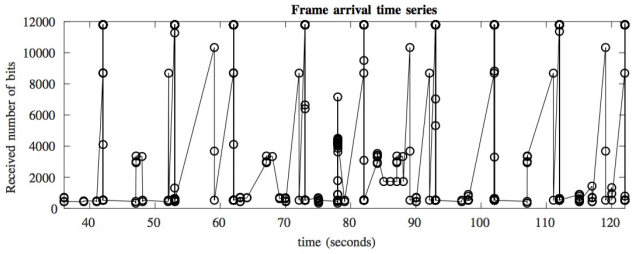


Fig. 2. Unevenly spaced time series observed when streaming audio to a client. The time and value pairs illustrated consist of frame arrival time-stamps and frame sizes. Every $\approx$ 10s a number of frames of size $\approx$ 12kbits are received. This time-series has a clear periodic component.

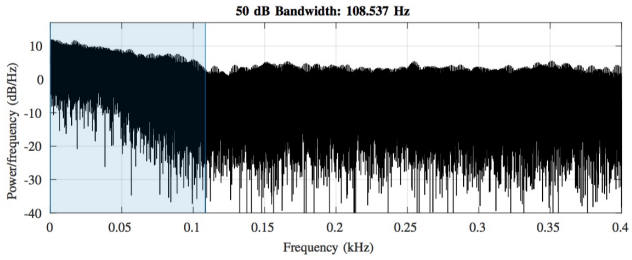


Fig. 3. PSD of unevenly spaced time-series observed when receiving a streamed podcast from SoundCloud. The 50dB bandwidth is 108Hz. The upper envelope of the PSD falls to 0dB at 100Hz.

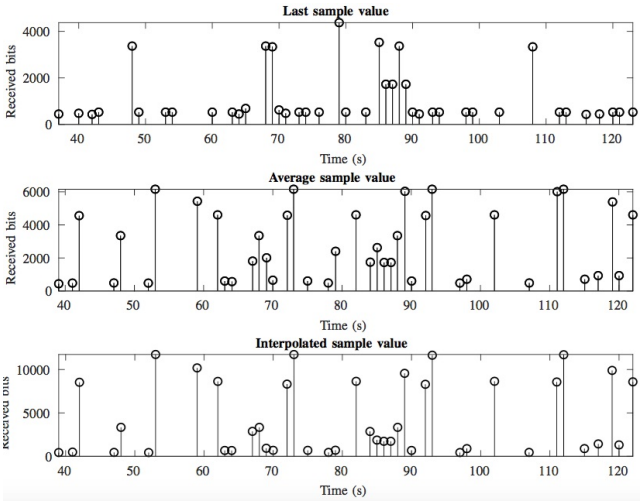


Fig. 4. Using periodic acquisition (time-step of 1 second), we illustrate the effect of using the last sample, average value and interpolated values at acquisition times $t \in \mathbb{N}$.

of each state acquisition value in Fig. 4; each trace should be compared with the original time series in Fig. 2.

Analysis: The PSD allows us to estimate the rate of change of the network/service – a parameter which is of crucial interest to service providers. For example, unusually high rates of change may indicate anomalies; periodic components may indicate that the network is healthy. The acquisition methods in Fig. 4 do not allow the network manager to estimate the

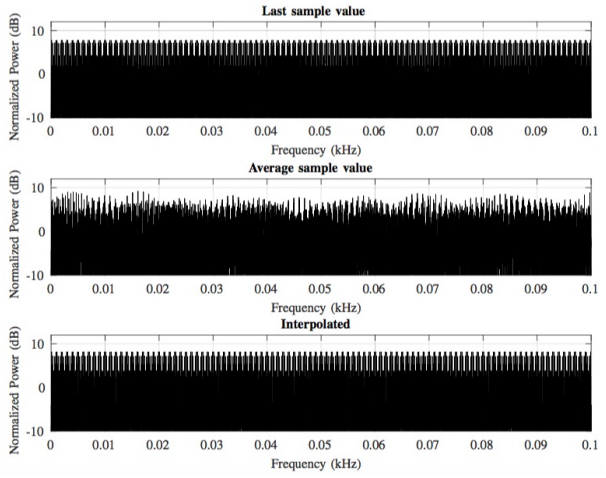


Fig. 5. PSD of current acquisition approaches: $x(p(t))$, $\mu_{t-1}(t)$ and $x_l(t)$. Important information such as the rate of change of the underlying time series and the period is lost. The flat PSDs illustrated demonstrate that higher frequency information has been lost.

maximum rate of change of the network. The information in the frequency band $0 < f < 108$Hz is lost by the acquisition methods used in Fig. 4. One reason for this is that they create an estimate for data that does not exist.

(1) Information about the period of the trace is lost. This is a serious shortcoming. For example an artifact of $x(p(t))$ is that its period is approximately 20 seconds and not approximately 10 seconds. Period calculations for each trace will be inaccurate as the times of frame arrivals in the original trace are never in the set of times $t \in \mathbb{N}$. For example the 12kb frame arrival times occur just after $40, 50, 60, \ldots$ seconds. The large frame arrivals occur just before $50, 70, \ldots$ seconds in $x(p(t))$. In the average acquisition trace $\mu_{t-1}(t)$, the pulse amplitudes (frame sizes) vary in height, which makes period detection hard. Finally the interpolated acquisition trace $x_l(t)$ exhibits pulse amplitudes which vary each time they appear with a period of 20 seconds. Estimates of the period tell us when to expect the next burst of podcast data. The ability to detect if this burst of data has arrived, or has arrived late, is lost. This is because the exact times when bursts of data arrive have been lost. Secondly, the sizes of the burst have been lost; this is due to averaging or interpolation, or because the previous observation was not part of the periodic train of frames. The effect of using the last, average or interpolated value over the previous second causes the period information to be obscured. Averaging unevenly spaced time series introduces ambiguity.

(2) Higher frequency information in the range up to 108Hz and above is lost in Fig 5. The acquisition techniques, $x(p(t))$, $\mu_{t-1}(t)$ and $x_l(t)$ low-pass filter the original unevenly spaced time-series. We posit that high-frequency variations in this trace may help us detect sub-optimal network performance. Choosing a low-pass filter in the acquisition step, without reference to the rate of change of the time series, may

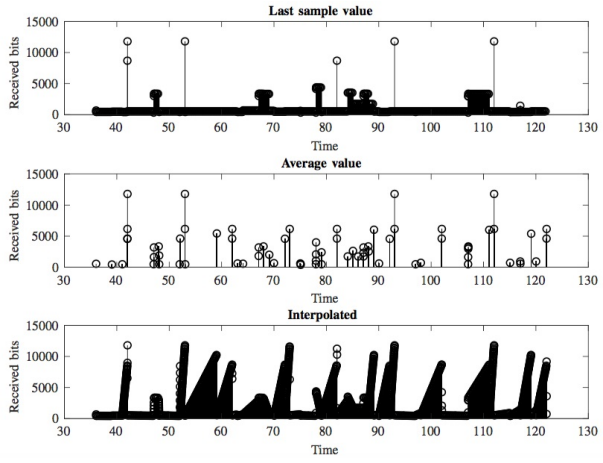


Fig. 6. High-rate periodic acquisition: Acquisition at 200 Hz does not significantly improve the ability to estimate the period and rate of change of the original data from the acquired time-series.

remove crucial monitoring and problem diagnosis information. **Remark:** the amplitude and location in time of the acquired stems (in Fig. 4) depend on the relative position of each of the events in x relative to the acquisition times $t \in \mathbb{N}$. Shifting the acquisition times relative to the events in x, can greatly increase or decrease the efficacy of acquisition and monitoring. We have shown that for an arbitrary periodic acquisition starting time that periodic acquisition is harmful for higher-order functions such as monitoring and learning.

Higher acquisition rate: The periodic acquisition methods above, with a period of 1Hz, remove crucially important information about the fundamental frequency and the rate of change of frame delivery in Fig 2. Loss of this information will reduce the effectiveness of higher-order functions such as monitoring and learning that consume the acquired performance data. The PSD of the original unevenly spaced time-series experiences a drop of 50dB at approximately 100Hz in Fig. 3. We consider the effect of acquiring this time series using a significantly higher acquisition rate of 200Hz to determine if this facilitates the capture of crucial parameters such as the period and rate of change of the underlying trace. In effect we are assuming that the Nyquist rate of this unevenly spaced time series is $\approx$ 200Hz.

Increasing the rate of State Acquisition by a factor of 200 increases the bandwidth of the monitoring protocol that consumes these metrics. Fig. 6 demonstrates that increasing the acquisition rate 200-fold does not improve our ability to estimate the period and rate of change of the time series. The acquisition methods $x(p(t)), \mu_s(t)$ and $x_l(t)$ suffer to similar problems as before; but they now consume more bandwidth and storage.

IV. Rate of Change Estimation

In this section we determine the appropriate way to acquire periodic performance traces from audio and video streams by estimating the rate of change, c, of the trace. Even if the resulting rate of acquisition is unfeasibly high, knowledge of this parameter can facilitate better design decisions. For example, we can low-pass filter the signal and use this filtered time series in learning algorithms for example, cognizant of the fact that we cannot make predictions outside of a certain range of frequencies. We appeal to classical evenly spaced sampling theory for guidance with the challenge of rate of change estimation for unevenly spaced time series.

The problem with unevenly spaced time series: The periodic time-series we observe for video and audio stream applications, x, consist of a sum of weighted and delayed Dirac pulses, $\delta(t)$, e.g.

$$\sum_{n=1}^{N} x(t_n)\delta(t - t_n). \tag{11}$$

When these pulses form an infinitely long sequence of evenly spaced pulses, spaced T seconds apart, and the pulses have identical heights, this is called a Dirac comb [23]. It is a known result in Signal Processing that the Fourier transform of a Dirac comb with period Ts produces another Dirac comb in the frequency domain with period $\frac{1}{T}$Hz. As these Dirac pulses are spaced by $\frac{1}{T}$Hz in the frequency domain, to avoid aliasing, we must ensure that the spectral content of the signal we wish to acquire is confined to a region of $c = \frac{1}{2T}$Hz, which is the maximum permissible rate-of-change of this evenly spaced time series.

Unlike the uniform case, the Fourier transform of the unevenly spaced time series in Eqn. 11 will generally not be a Dirac Comb, that is, a sequence of uniformly spaced delta functions; the symmetry in the Dirac comb is broken by uneven spacing of the pulses, which leads to information rich transform we observe in Fig. 3. This is because in the Fourier domain, the locations and heights of the Dirac delta functions are related to the intervals between the time domain observations. Randomizing the observation times, randomizes the locations and heights of the Fourier domain peaks and heights.

To leverage the results of classical sampling [23] to estimate c for an unevenly spaced time series, we express a unevenly spaced time series as a evenly spaced time series. To this end, we determine the Dirac comb which has Dirac deltas that align exactly with each of the pulses in the unevenly spaced time series. In other words, we need to determine the largest T_a such that each $t_n \in T(x)$ can be written as $t_n = \alpha + nT_a$, for integers n and an arbitrary time offset α.

Definition (1) Accurate acquisition: To accurately acquire an unevenly spaced time series the acquisition period T_a should satisfy the condition

$$\Delta t(x) \mod T_a = 0, \forall t_n \in \mathbb{T}_N \tag{12}$$

Definition (2) Rate-of-Change: The rate of change of the unevenly spaced time series x is $c = \frac{1}{2T_a}$ if T_a satisfies the condition

$$\Delta t(x) \mod T_a = 0, \forall t_n \in \mathbb{T}_N \tag{13}$$

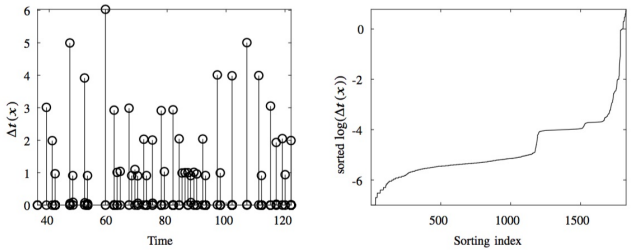


Fig. 7. The time durations between consecutive frame arrivals $\Delta t(x)$ are plotted (stem height) on the LHS versus experiment time (x-axis). These durations are sorted from smallest to largest on the RHS. Very few of the consecutive arrival times exhibit a large delay.

Finding the value of T_a that ensures that acquisition is being performed at a sufficiently high rate, leads to acquisitions rates which are impractical for real-world applications. If observation times are recorded to d decimal places, the largest rate of change is determined by machine precision.

To make progress, we use a rough upper bound on T_a, which consists of examining the minimum separation between consecutive frame arrivals:

$$\min \Delta t(x). \tag{14}$$

As the traces we will consider are generally base-band signals, we desire a value of T_a which captures most of the spectral energy of the trace x.

Analysis: The smallest difference between two time-stamps, $\min \Delta t(x)$ in Fig. 2 is 10^{-7} seconds. This would necessitate acquiring the signal at an unfeasibly high rate, $> 10^7$ Hz, which is impractical most networking applications. We propose a Greedy Acquisition solution to solve this problem and examine a trade-off between the accuracy of the acquired time-series, the bandwidth of different versions of the acquired time-series.

V. Greedy Acquisition

A Greedy Acquisition algorithm for unevenly spaced time-series is presented that acquires observations at a rate which is consistent with the rate of change of the original time series, c.

Fig. 7 demonstrates the number of time intervals between consecutive values in x (the audio trace) that are greater than 1 second (on the RHS). In summary, only 1.5% of these time intervals are greater than 1s; the majority of the time spacings between events are much smaller. A useful rule of thumb for periodic acquisition is that if we increase the time between acquisitions in the acquisition routine, we decrease the range of frequencies that we can observe in the trace [23]. It is challenging to perform accurate acquisition using a periodic acquisition scheme because the time-step must be sufficiently small to capture a range of rates of change. Using a fixed, periodic acquisition rate which is of the order of 1Hz (in [5] and [7]) or even one kilohertz (in [6]) will not yield an accurate representation of the performance of x. For example, for the audio streaming trace in Fig. 2, an acquisition rate of 1Hz is significantly too low an acquisition rate for 98.5% of the frame arrivals in this time-series. Averaging these values over a one-second interval, removes all frequency components above $\approx$ 1Hz, which make subsequent functions such as problem diagnosis difficult. Similarly in the case of fixed periodic acquisition rate of 1kHz (which is the time-step used by Prometheus [8]) this rate is too low for 94.25% of the frame arrivals in Fig. 2.

From inspection of Fig. 7 it is clear that locally in time, the value of $\min \Delta t(x)$ may be much higher than it would for the entire time series. This means that an acquisition performed at a rate that is consistent with the local rate of change of the time series, could be expected to significantly reduce the bandwidth required to accurately represent the underlying event steam x, over a periodic acquisition scheme with the same level of fidelity. We investigate an adaptive acquisition protocol, where the value of T_a changes with time, and posit that it should give a more accurate time series representation, using less bandwidth than before.

A greedy approximation for the time series x is generated by summing a finite number of functions g_i taken from a dictionary of such functions D. For example, the function g_1 consists of a sum of Diracs which are weighted by the largest value of the time series x, e.g. $m_1 = \max(V(x))$, which gives the function

$$g_1 = m_1 \sum_k \delta(t - t_1(k)) \tag{15}$$

where t_1 is the set of times where $x = m_1$ and there are k elements in this set.

The next function in the dictionary D is g_2. It consists of a sum of Diracs which are weighted by the second largest value of the time series x, $m_2 = \max(V(x) \setminus m_1)$. In words, we remove all of the instances of the maximum value of x from x and then find the next largest value, m_2. We then construct the function g_2 which has Dirac pulses of height m_2 every time $x = m_2$,

$$g_2 = m_2 \sum_k \delta(t - t_2(k)). \tag{16}$$

Similar to the previous case, t_2 is the set of times where $x = m_2$; there are k elements in this set. Continuing this process, we construct the entire dictionary $D = \{g_i\}$.

A j-th order approximation of x, which is denoted y_j, is obtained by summing up the first j elements of the dictionary

$$y_j := \sum_{i=1}^{j} g_i. \tag{17}$$

The accuracy of the jth order approximation is computed using the Frobenius norm,

$$\epsilon_j = \sqrt{\int (x - y_j)^2 dt}, \tag{18}$$

where $y_j(t_n) = 0$ if we have not acquired a value at time $t = t_n$ for the jth approximation.

The set of values $m_1, m_2, \ldots$ are the values of the frame sizes sorted from largest to smallest. Because these frames are generated as part of well-defined protocols, which have

Data: input: real-time process for acquisition from x and β.
Result: acquired time series (t_n, y_n).
initialization: sorted list of frame sizes m_i from historical data;
while *still receiving stream* **do**
  read current value;
  if $x > \beta$ **then**
    acquire time $t_n = t$;
    acquire the value $y_n = x_n$;
  else
    do not acquire value or time;
  end
end

Algorithm 1: Online Greedy Acquisition Algorithm

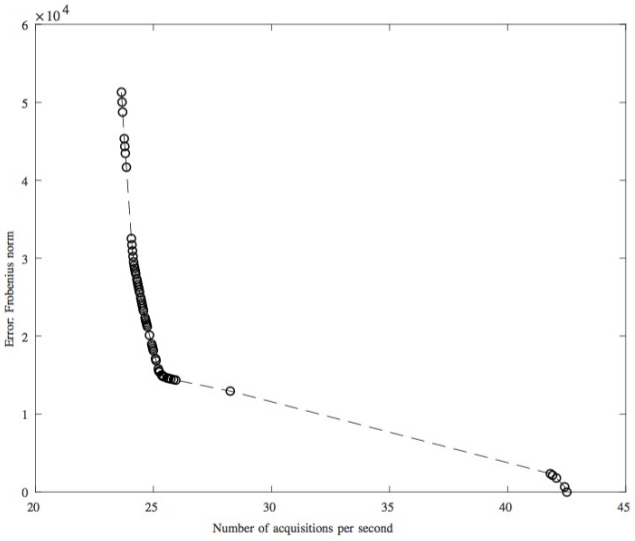


Fig. 8. Frame arrival process statistic acquisition for the SoundCloud trace using Greedy Acquisition: The error in the acquisition time-series decreases as the number of acquisitions per second increases. We examine the first through to the sixtieth order approximation of x. The error achieved when 25 measurements are acquired on average per second is impressively low, given that we are computing the Frobenius norm of frame sizes of up to 12kbits.

predetermined frame sizes for an array of scenarios, the number of different values of m_i that we can expect is finite, and generally, quite small. It is possible to determine the set $\{m_i\}_i$ a-priori for any given service.

An online real-time version of the Greedy Acquisition algorithm presented above, is presented in Alg. 1. It consists of determining whether or not the current frame arrival has a frame size which is larger than a user defined threshold, β. The value of β determines the accuracy of the achieved approximation. If an incoming frame size is larger than β, the frame size and time are recorded as part of the unevenly spaced acquired time series (t_n, y_n). Greedy approximation has the following properties. (1) the accuracy of the approximation increases monotonically as β decreases, because each increase in the order of the approximation j, reduces the error in

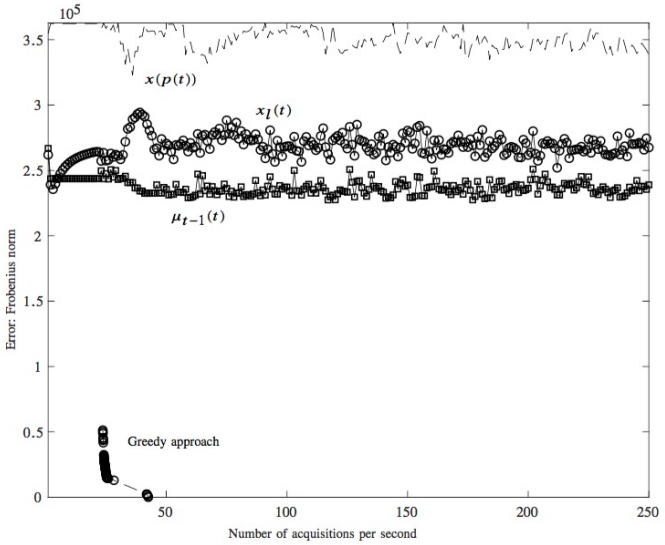


Fig. 9. Comparison of the acquisition accuracy bandwidth trade-off achieved using tradition periodic acquisition methods (last observation, average and interpolation) and using Greedy Acquisition. The greedy approach is significantly better.

the approximation of x. (2) The bandwidth requirement of greedy approximation increases as j increases, but it increases efficiently, in line with the rate-of-change of x. (3) Because the original frame arrival times are used, the periodic component of the time series is generally captured by (t_n, y_n); moreover, the rate-of-change of the time-series (t_n, y_n) is generally consistent with that of (t_n, x_n), in comparison with a periodic acquisition algorithm, which chooses the acquisition period in an arbitrary way, and is thus, not always suited to the process being observed.

VI. Numerical Evaluation

We examine the accuracy of different acquisition methods using a simple video and audio use-case. We focus on audio and video because according to Cisco, by 2021, 82% of all consumer Internet traffic will be IP video traffic [24]. The ability to acquire measurements from the traces clients receive will be crucial. Higher quality acquisition data will improve next-generation monitoring protocols.

Our hypothesis is that Greedy Acquisition yields more accurate estimates of the observed process than traditional periodic monitoring approaches. Moreover, we posit that Greedy Acquisition uses less bandwidth and storage, and that the acquired time-series preserve important statistical features of the observed trace, such as its period. In our comparison we implement acquisition functions which estimate $x(p(t))$, $\mu_s(t)$ and $x_I(t)$. The reason that we do not use off-the-shelf tools such as SAR [5] is that they have a minimum acquisition rate of 1Hz (in the case of SAR), a rate which we would like to significantly increase, in order to fully evaluate the potential of periodic acquisition. We consider a frame arrival process because TCPDUMP [22] provides easy access to high

resolution time-stamps for a widely available unevenly spaced time series, which is representative of what we might observe at many other intermediate points in the network.

Experimental set-up: In the first scenario a client streams a podcast from an online audio distribution platform, a SoundCloud instance [14]. In 2014 SoundCloud boasted 75 million unique monthly listeners, which demonstrates the popularity of the service, and motivates the need to be able to accurately acquire measurements from the audio traces. In the second scenario a client streams a video from a video server of an Irish public service broadcaster [25].

During our evaluation of both scenarios, the client requested the service and then TCPDUMP [22] was used to capture a description of the contents of the frames received on the client's network interface. Once the media session had terminated, we converted the captured .pcap file to text format using *tshark*, and we *grep*ed the resulting text file for the frame arrival times, t_n, and frame sizes x_n, forming the unevenly spaced time series (t_n, x_n). Frame arrival times were captured in terms of hours, minutes, seconds, and fractions of a second since midnight. We recorded the size of received frames using the "bytes on wire" field, in bits. We stored frame arrival times using double-precision according to IEEE Standard 754 for double precision, which bounded the precision of rate-of-change estimates. The frame sizes were integer-valued. For the SoundCloud session, the maximum frame received was 11792 bits, and 60 different frame sizes were observed during this session, which gave us 60 different potential acquisition accuracies. In all cases we streamed data for ≈ 15 minutes.

Discussion: Fig. 8 illustrates the error (Eqn. 18) versus bandwidth trade-off achieved for a SoundCloud session by varying β in the online Greedy Acquisition algorithm. As the value of β is decreased, the accuracy of the greedy representation improves. This accuracy comes at the cost of 25 acquisitions per second on average.

This result is significant, particularly when it is compared with the periodic, last-observation, averaging and interpolation schemes currently used ($x(p(t)), \mu_s(t)$ and $x_l(t)$). Fig. 9 illustrates the accuracy of $x(p(t))$, $\mu_s(t)$ and $x_l(t)$ as a function of the bandwidth consumed by these acquisition approaches. The bandwidth is increased by increasing the resolution of the acquisition time-step. For the periodic acquisition methods, the trend is for the error to decrease as the rate of acquisition increases. We increase the rate of acquisition from 1Hz up to 250Hz in steps of 1Hz.

In this trace the times of the arrivals of the largest frame sizes tend to determine the periodic component of the time series. Preserving the exact times of events is important. The ability to so, is determined by the rate of acquisition. The Greedy Acquisition algorithm acquires a trace at a rate which is consistent with the rate of change of time series. For the 1st approximation y_1, the minimum time-step between consecutive frame arrivals $\min \Delta t(y_1)$ does not change as we increase the order of the approximation. The 1st approximation y_1, and all subsequent approximations have the possibility of capturing the fastest changes in the observed trace, depending on their importance (frame size).

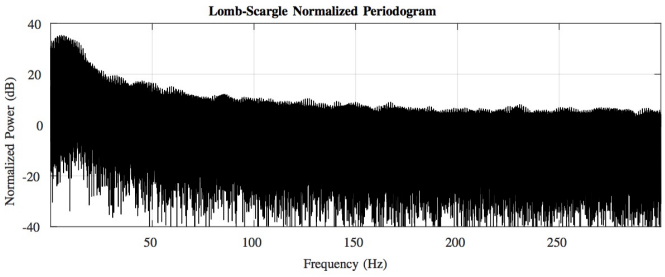


Fig. 10. PSD of frame arrival time series for video streaming session. This trace has a periodic component at 5.4Hz, which implies that acquiring this trace at 1Hz is insufficient.

The traditional periodic approaches have time steps in the range $1/p$s where $p = 1 \ldots 250$Hz in these experiments. These periodic approaches cannot observe the fastest changes in the observed process, or at the maximum rate of change of the process. In conclusion, periodic acquisition approaches yield an error which is 5 times worse than the error of Greedy Acquisition, using 5 times the bandwidth to do so. Finally, important statistics of the observed traces are typically not preserved by periodic approaches.

In the case of the video streaming session, we illustrate the PSD of the time-series in order to provide initial estimates for the period and the rate-of-change of x. This trace has a periodic component at 5.4Hz, which implies that acquiring this trace using a periodic acquisition method at 1Hz is insufficiently accurate. Note that even the recommended reporting rate of RTCP (5Hz) is not sufficiently large to capture the period of this trace. In addition, the PSD of this trace exhibits significant power up to 100Hz. Similar to the audio streaming scenario, increasing the acquisition rate of periodic acquisition algorithms does not significantly improve the accuracy of the acquired time series.

With regard to the rate of change of this trace, 0.16% of consecutive frame arrivals have an inter-arrival time of greater than 1s, and 1.3% have an inter-arrival time of greater than 10^{-3}s. The minimum inter-arrival time is less than 10^{-7}s. Given the definition of the rate of change of unevenly spaced times series, periodic acquisition at 1Hz is insufficient.

These statistics underline the difficulty of choosing a time-step for periodic acquisition that would yield sufficiently high accuracy. We evaluate Greedy Acquisition, to see if acquiring measurements at a rate which is consistent with the rate of change of the trace, is accurate. Fig. 11 demonstrates that accurate acquisition is achieved by taking 300 acquisitions per second using Greedy Acquisition. The average acquisition rate for video that achieves the same accuracy as audio acquisition is approximately $\times 10$ the acquisition rate for audio. Once again Greedy Acquisition out-performs each of the periodic acquisition approaches – by an order of magnitude drop in the error of the representation – when 300 acquisitions are made per second.

Recommendations: Many current periodic acquisition ap-

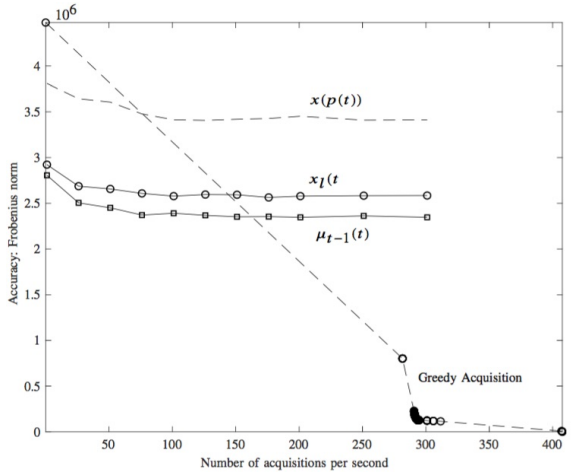


Fig. 11. Frame arrival process statistic acquisition for video trace using Greedy Acquisition: The error in the acquisition time series decreases as the number of acquisitions per second increases. The error achieved when 300 measurements are acquired on average per second is impressively low. The periodic acquisition methods $x(p(t))$, $\mu_{-1}(t)$ and $x_l(t)$ are illustrated for completeness.

proaches acquire time-series without knowledge of the underlying rate of change of the time series under observation. We have argued that knowledge of the rate of change of the process under observation, should drive the process of deciding when to acquire measurements of this process. Many networking time-series exhibit properties such as periodicity. These properties should be preserved by acquisition routines. One point of note from this work is that the times of events $T(x)$ are as important as the values recorded for these events $V(x)$. Periodic acquisition routines such as Nagios, record values with greater precision than times, due the default time resolution of 10^{-3}s. A second point of note is that the wide array of off-the-shelf acquisition routines means that little research is being done in the area. The received wisdom is that metric acquisition routines exist, and thus, there is little point in re-inventing them. Finally, we have provided evidence that Greedy Acquisition gives improved acquisition performance.

VII. Conclusions

In this paper we showed that acquiring a system or service's state at a rate which was consistent with the rate of change of the system (or service) provided a high-quality record of the state of the system. Research on capturing system state has lagged behind the growth of networks and the applications that use these networks as a substrate. Today, many monitoring and learning solutions rely on standard periodic State Acquisition solutions, which acquire the system state at a frequency of 1Hz. These solutions do not capture important characteristics of the signals they acquire, for example, periodicity and rate of change. For periodic traces, the ability to estimate the period from an acquired representation of the trace is fundamental. We demonstrated that the rate of change of many applications is much greater than 1 Hz, and then, we demonstrated that present-day acquisition techniques do not capture information which is crucially important for problem diagnosis. Our experiments with real-world voice and video traces demonstrate that high quality State Acquisition is possible if the time-stamps and magnitudes of events are recorded at the rate of change of the application.

References

[1] D. Jurca and R. Stadler, "H-GAP: estimating histograms of local variables with accuracy objectives for distributed real-time monitoring," *IEEE Trans. Net. and Serv. Man.*, vol. 7, no. 2, pp. 83–95, Jun. 2010.

[2] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "Predicting real-time service-level metrics from device statistics," *IFIP/IEEE Int. Sym. Int. Net. Man.*, pp. 1–8, 2015.

[3] R. de Fréin, "Source separation approach to video quality prediction in computer networks," *IEEE Comm Ltr*, vol. 20, no. 7, pp. 1333–1336, Jul. 2016.

[4] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Trans Net. Serv. Man.*, vol. 14, no. 1, pp. 106–120, 2017.

[5] S. Godard, "SAR," http://linux.die.net/man/1/sar.

[6] E. Galstad. (1999) Nagios. [Online]. Available: https://www.nagios.com/

[7] R. Binns. top. [Online]. Available: https://linux.die.net/man/1/top

[8] J. Volz and B. Rabenstein. (2018) Sound Cloud Developers: Backstage Blog. [Online]. Available: https://developers.soundcloud.com/blog/prometheus-monitoring-at-soundcloud

[9] ETSY. StatsD. [Online]. Available: https://github.com/etsy/statsd

[10] R. de Fréin, C. Olariu, Y. Song, R. Brennan, P. McDonagh, A. Hava, C. Thorpe, J. Murphy, L. Murphy, and P. French, "Integration of QoS Metrics, Rules and Semantic Uplift for Advanced IPTV Monitoring," *J. Net. Sys. Man.*, vol. 23, no. 3, pp. 673–708, Jul 2015.

[11] R. de Fréin, "The data-centre whisperer: Relative attribute usage estimation for cloud servers," in *24th EUSIPCO*, Aug. 2016, pp. 687–691.

[12] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. of Int. Serv. and Apps*, vol. 1, no. 1, pp. 7–18, May 2010.

[13] D. Josephsen, *Building a Monitoring Infrastructure with Nagios*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[14] A. Ljung and E. Wahlforss. (2008) SoundCloud. [Online]. Available: https://soundcloud.com/

[15] R. de Fréin, "Effect of system load on video service metrics," in *IEEE Irish Sig. Sys. Conf. (ISSC)*, 2015, pp. 1–6.

[16] "The Cacti Group Inc." [Noline], 2016, accessed on Dec. 28, 2016. [Online]. Available: http://www.cacti.net/

[17] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne, "RTP: A transport protocol for real-time applications," IETF RFC3550, 2003. [Online]. Available: https://tools.ietf.org/html/rfc3550

[18] A. Begen, T. Akgul, and M. Baugher, "Watching video over the web: Part 1: Streaming protocols," *IEEE Int. Comp.*, vol. 15, no. 2, pp. 54–63, Mar. 2011.

[19] ——, "Watching video over the web: Part 2: Applications, standardization, and open issues," *IEEE Int. Comp.*, vol. 15, no. 3, pp. 59–63, 2011.

[20] "Datadog," Online documentation, accessed on Dec. 28, 2016. [Online]. Available: https://www.datadoghq.com/

[21] N. R. Lomb, "Least-squares frequency analysis of unequally spaced data," *Astrophy. and Space Sc.*, vol. 39, no. 2, pp. 447–462, Feb. 1976.

[22] V. Jacobson, C. Leres, and S. McCanne. tcpdump. [Online]. Available: http://www.tcpdump.org/

[23] M. Vetterli, J. Kovacevic, and V. K. Goyal, *Foundations of Signal Processing*. Cambridge: Cambridge Univ. Press, 2014.

[24] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[25] (1996) TG4: TG Ceathair. [Online]. Available: http://www.tg4.ie/en/player/home/

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under the Grant Number 15/SIRG/3459.

Towards a Deeper Understanding of TCP BBR Congestion Control

Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle
Chair of Network Architectures and Services, Technical University of Munich
{scholzd|jaeger|schwaigh|raumer|fgeyer|carle}@net.in.tum.de

***Abstract*—In 2016, Google published the bottleneck bandwidth and round-trip time (BBR) congestion control algorithm. Unlike established loss- or delay-based algorithms like CUBIC or Vegas, BBR claims to operate without creating packet loss or filling buffers. Because of these prospects and promising initial performance results, BBR has gained wide-spread attention. As such it has been subject to behavior and performance analysis, which confirmed the results, but also revealed critical flaws.**

Because BBR is still work in progress, measurement results have limited validity for the future. In this paper we present our publicly available framework for reproducible TCP measurements based on network emulation. In a case study, we analyze the TCP BBR algorithm, reproduce and confirm weaknesses of the current BBR implementation, and provide further insights. We also contribute an analysis of BBR's inter-flow synchronization behavior, showing that it reaches fairness equilibrium for long lived flows.

***Index Terms*—TCP, Congestion Control, BBR, Reproducible Measurements**

I. Introduction

TCP BBR is a congestion-based congestion control algorithm developed by Google and published in late 2016 [1]. In contrast to traditional algorithms like CUBIC [2] that rely on loss as indicator for congestion, BBR periodically estimates the available bandwidth and minimal round-trip time (RTT). In theory, it can operate at Kleinrock's optimal operating point [3] of maximum delivery rate with minimal congestion. This prevents the creation of queues, keeping the delay minimal.

Service providers can deploy BBR rapidly on the sender side, as there is no need for client support or intermediate network devices [1]. Google already deployed BBR in its own production platforms like the B4 wide-area network and YouTube to develop and evaluate BBR [1] and provided quick integration of BBR with the Linux kernel (available since version 4.9). This spiked huge interest about benefits, drawbacks and interaction of BBR with alternatives like CUBIC. The research community has started to formalize and analyze the behavior of BBR in more detail. While the initial results published by Google have been reproducible, demonstrating that BBR significantly improved the bandwidth and median RTT in their use cases, weaknesses like RTT or inter-protocol unfairness have been discovered since (e.g. [4, 5, 6]). As a consequence, BBR is actively improved [5]. Proposed changes usually aim to mitigate specific issues, however they need to be carefully studied for unintended side effects.

ISBN 978-3-903176-08-9

We introduce a framework for automated and reproducible measurements of TCP congestion avoidance algorithms. It produces repeatable experiments and is available as open source at [7]. The use of emulation using Mininet allows the framework to be independent of hardware, enabling other research groups to easily adapt it to run their own measurements or replicate ours.

We demonstrate the capabilities of our framework by inspecting and analyzing the behavior of BBR in different scenarios. While the throughput used for our measurements is orders of magnitude lower compared to testbeds utilizing hardware, we verify the applicability of our results by reproducing measurements of related work. Beyond reproduction, we deepen the analysis of BBR regarding inter-flow unfairness and inter-protocol fairness when competing with TCP CUBIC flows. Lastly, we use measurements to analyze the inter-flow synchronization behavior of BBR flows.

This paper is structured as follows: Section II presents background to TCP congestion control. In Section III, we describe our framework for reproducible TCP measurements. We performed various case studies with the analysis of BBR. The results are used to validate our framework by reproducing and extending measurements from related work in Section IV. Our BBR inter-flow synchronization analysis is discussed in Section V. Related work is presented in Section VI before we conclude with Section VII.

II. TCP Congestion Control

Congestion control is required to achieve high network utilization for multiple flows, claiming a fair share, while preventing overloading the network with more packets than can be handled. Buffers are added to counteract packet drops caused by short lived traffic peaks, increasing network utilization. When buffers remain non-empty ("static buffers"), they add delay to every packet passing through the buffer, coined *bufferbloat*. Static buffers originate mainly from two factors, as shown by Gettys and Nichols [8]: poor queue management and failure of TCP congestion control. Algorithms like TCP NewReno [9] or TCP CUBIC [2] use packet loss as indication of congestion. However loss only occurs when the buffers are close to full at the bottleneck (depending on the queue management used). The congestion is only detected when the bottleneck is already overloaded, leading to large delays hurting interactive applications.

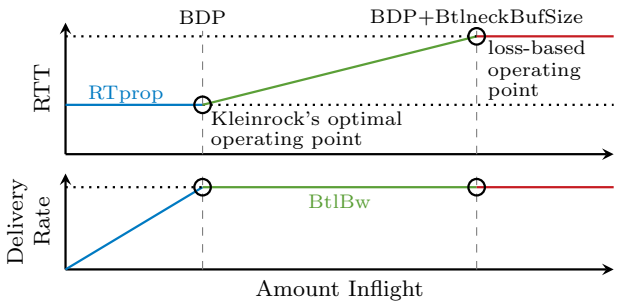


Figure 1: Effect of increasing inflight data on the RTT and delivery rate. Based on [1].

Various TCP congestion control algorithms were developed to improve on loss-based congestion control. Examples include TCP Vegas [10], adapting delay as indicator, or TIMELY [11] based on precise RTT measurements. However, these are suppressed when competing with loss-based algorithms. Hock et al. present TCP LoLa [12], primarily focusing on low latency. Hybrid algorithms using both loss and delay as congestion indication were proposed such as TCP Compound [13]. Alizadeh et al. proposed Data Center TCP (DCTCP) [14], which requires support for Explicit Congestion Notification (ECN) in network switches.

A. TCP Optimal Operation Point

Any network throughput is limited by the segment with the lowest available bandwidth on the path. It is called bottleneck, as it limits the total throughput of the connection. Thus for modeling congestion control, a complex network path can be modeled by a single link. The delay of that link is set to the sum of all propagation delays in each direction and the bandwidth is set to the bottleneck's (BtlBw). This preserves the round trip propagation delay (RTprop). The bandwidth-delay product (BDP) as $\text{BtlBw} \cdot \text{RTprop}$ describes the amount of data that can be inflight (non-acknowledged) to fully utilize the network path and is coined Kleinrock's optimal point of operation [3].

Figure 1 visualizes the effects of an increase in inflight data on the connection's bandwidth and RTT. If less data than the BDP is inflight, there is no congestion and the RTT equals RTprop (application bound). The delivery rate corresponds directly to the sending rate, but hits the maximum when the inflight data reaches the BDP at Kleinrock's point. Increasing the inflight further causes packets to arrive faster at the bottleneck than they can be forwarded. This fills a queue, causing added delay which increases linearly with the amount inflight (recognized by delay-based algorithms). The queue is full when the amount inflight hits $\text{BDP} + \text{BtlneckBufSize}$. After this point, the bottleneck buffer starts to discard packets (recognized by loss-based algorithms), capping the RTT. This shows that both delay and loss-based algorithms operate beyond Kleinrock's optimal operating point.

B. Bottleneck Bandwidth and Round-trip Propagation Time

The following describes basics of BBR that are important for our evaluation. Our deliberations are based on the version presented by Cardwell et al. [1] and we refer to their work for a detailed description of the congestion control algorithm or [4] for a formal analysis.

1) Overview: The main objective of BBR is to ensure that the bottleneck remains saturated but not congested, resulting in maximum throughput with minimal delay. Therefore, BBR estimates bandwidth as maximum observed delivery rate BtlBw and propagation delay RTprop as minimum observed RTT over certain intervals. Both values cannot be measured simultaneously, as probing for more bandwidth increases the delay through the creation of a queue at the bottleneck and vice-versa. Consequently, they are measured separately.

To control the amount of data sent, BBR uses *pacing gain*. This parameter, most of the time set to one, is multiplied with BtlBw to represent the actual sending rate.

2) Phases: The BBR algorithm has four different phases [15]: Startup, Drain, Probe Bandwidth, and Probe RTT.

The first phase adapts the exponential **Startup** behavior from CUBIC by doubling the sending rate with each round-trip. Once the measured bandwidth does not increase further, BBR assumes to have reached the bottleneck bandwidth. Since this observation is delayed by one RTT, a queue was already created at the bottleneck. BBR tries to **Drain** it by temporarily reducing the pacing gain. Afterwards, BBR enters the **Probe Bandwidth** phase in which it probes for more available bandwidth. This is performed in eight cycles, each lasting RTprop: First, pacing gain is set to 1.25, probing for more bandwidth, followed by 0.75 to drain created queues. For the remaining six cycles BBR sets the pacing gain to 1. BBR continuously samples the bandwidth and uses the maximum as BtlBw estimator, whereby values are valid for the timespan of ten RTprop. After not measuring a new RTprop value for ten seconds, BBR stops probing for bandwidth and enters the **Probe RTT** phase. During this phase the bandwidth is reduced to four packets to drain any possible queue and get a real estimation of the RTT. This phase is kept for 200 ms plus one RTT. If a new minimum value is measured, RTprop is updated and valid for ten seconds.

III. TCP Measurement Framework

The development of our framework followed four requirements. **Flexibility** of the framework should allow to analyze aspects of TCP congestion control, focusing on but not limited to BBR. The **Portability** of our framework shall not be restricted to a specific hardware setup. **Reproducibility** of results obtained via the framework must be ensured. Given a configuration of an experiment, the experiment itself shall be repeatable. All important configuration parameters and the results should be gathered to allow replicability and reproducibility by others. The complete measurement process shall be simplified through **Automation**. Via configuration files and experiment description, including post processing of data and

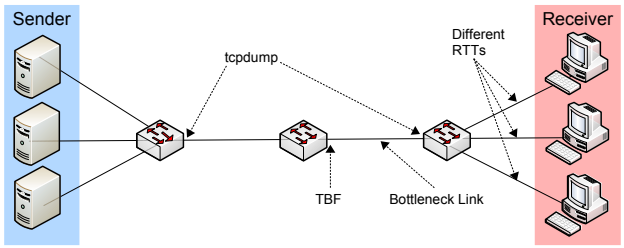


Figure 2: Mininet setup with sending and receiving hosts and bottleneck link.

generation of plots, the experiment should be executed without further user interaction.

A. Emulation Environment

Our framework uses emulation based on Linux network namespaces with Mininet. Linux network namespaces provide lightweight network emulation, including processes, to run hundreds of nodes on a single PC [16]. A drawback is that the whole system is limited by the hardware resources of a single computer. Thus we use low bandwidths of 10 Mbit/s for the different links in the studied topology. By showing in Section IV that our measurements yield similar results as related work performing measurements beyond 10 Gbit/s, we argue that the difference in throughput does not affect the validity of the results.

B. Setup

Topology: As a TCP connection can be reduced to the bottleneck link (cf. Section II-A), our setup uses a dumbbell topology depicted in Figure 2. For each TCP flow a new host-pair, sender and receiver, is added for simplified collection of per-flow data. Both sides are connected via three switches. The middle switch acts as the bottleneck by performing traffic policing on its interface. The two additional switches allow capturing the traffic before and after the policing. Traffic from the receivers to the senders is not subject to rate limiting since we only send data from the senders and the returning acknowledgment stream does not exceed the bottleneck bandwidth, assuming symmetric bottleneck bandwidth.

Delay Emulation: We use NetEm to add flow specific delay at the links between the switch and the respective receivers to allow configurable RTTs. This approach introduces problems for higher data rates like 10 Gbit/s where side effects (e.g. jitter) occur [4], but works well for the data rates we use.

Rate Limit & Buffer Size: We use Linux's Token-Bucket Filter (TBF) for rate limiting and setting the buffer size. TBFs also allow a configurable amount of tokens to accumulate when they are not needed and the configured rate can be exceeded until they are spent. We set this *token bucket size* to only hold a single packet, because exceeding the bottleneck bandwidth even for a short time interferes with BBRs ability to estimate the bottleneck bandwidth correctly [1].

C. Workflow

Each experiment is controlled using a configuration file describing the flows. For each flow, the desired TCP congestion control algorithm, start time in relation to previous flow, RTT, and runtime have to be specified. The runtime of an experiment consists of a negligible period to set up Mininet, as well as the actual experiment defined by the length of the running flows. The framework automatically extracts data and computes the implemented metrics.

D. Metric Collection

For each TCP flow we gather the sending rate, throughput, current RTT, and the internal BBR values. We also sample the buffer backlog of the TBF every 40 ms. As a result of one experiment, a report containing 14 graphs visualizing the metrics over time is automatically generated. A sample experiment report, including its configuration file, can be found with our source code publication [7].

We capture the headers up to the TCP layer of all packets before and after the bottleneck using `tcpdump`. The raw data is processed to generate the metrics listed below. Existing tools like Wireshark (including the command line tool `tshark`) and `tcptrace` did not meet all our requirements for flexibility. Instead we wrote our own analysis program in Python.

Sending Rate & Throughput: We compute the per flow and total aggregated sending rate as the average bit-rate based on the IP packet size in 200 ms intervals, using the capture before the bottleneck. The throughput is computed equal to the sending rate, but is based on the capture after the bottleneck to observe the effect of the traffic policing.

Fairness: We follow the recommendation of RFC 5166 [17] and use Jain's Index [18] as fairness coefficient based on the sending rate to indicate how fair the bandwidth is shared between all flows. For n flows, each of them allocating $x_i \geq 0$ of a resource,

$$\mathcal{F} = 1/n \cdot [\Sigma_{i=1}^{n} x_i]^2 / \Sigma_{i=1}^{n} x_i^2$$

is 1 if all flows receive the same bandwidth and $1/n$ if one flow uses the entire bandwidth while the other flows receive nothing. The index allows quantifying the fairness in different network setups independent of the number of flows or the bottleneck bandwidth. Graphs displaying the fairness index in the remaining part of this paper are restricted to the interval $[1/n, 1]$ unless mentioned otherwise.

Round-trip Time: RTT values are aggregated in intervals of 200 ms and averaged to provide better stability. Samples of retransmitted packets are ignored.

Retransmissions: We count retransmissions of TCP segments in the packet capture before the bottleneck. We use these as an indicator for packet loss in our evaluation.

Inflight Data: Refers to the number of bytes sent but not yet acknowledged. We obtain this value by computing the difference of the maximum observed sequence and acknowledgment numbers in the capture before the bottleneck. This metric is only useful when there are no retransmissions.

BBR Internal Values: BBR keeps track of the estimated bottleneck bandwidth and RTT as well as the pacing and window gain factors. We extract these values every 20 ms using the `ss` tool from the `iproute2` tools collection.

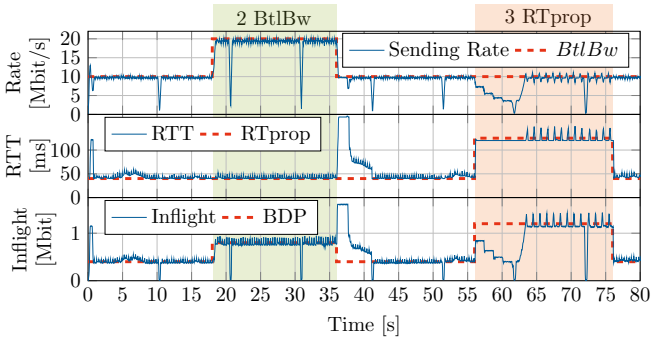


Figure 3: Single BBR flow (40 ms, 10 Mbit/s bottleneck) under changing network conditions. Values sampled every 40 ms.

E. Limitations

Due to resource restriction on a single host emulated network, we are limited to bandwidths in the 10 Mbit/s range. However, our methodology provides sufficient accuracy compared to measurements utilizing real hardware. This is because both approaches use the same network stack, i.e., the same implementation of the BBR algorithm.

The CPU and RAM capacities of the test system limit the number of emulated hosts and therefore flows. We encountered problems when spawning more than 30 hosts simultaneously using a host equipped with an Intel Core i5-2520M CPU @ 2.50 GHz and 8 GB RAM.

IV. Reproduction & Extension of Related Work

We validate the accuracy of our framework by using it to reproduce the results of related work that were based on measurements with hardware devices. The results show that the behavior of TCP BBR at bandwidths in the Mbit/s range is comparable to the behavior at higher ranges of Gbit/s. In the following, we present our reproduced results with a mention of the respective related work.

We focus on the results of two research groups. Cardwell et al., the original authors of BBR, have described their current research efforts towards BBR 2.0 [1, 5]. Goals are reduced loss rate in shallow buffers, reduced queuing delay and improved fairness among others. Hock et al. evaluated BBR in an experimental setup with 10 Gbit/s links and software-based switches [4]. They reproduced intended behavior of BBR with single flows, but also showed cases with multiple flows where BBR causes large buffer utilization.

For all following figures the raw data, post-processed data and source code to generate the figures can be found with our source code publication [7]. Unless representing a single flow, measurements were repeated five times and standard deviations are shown where applicable.

A. Single Flow

Figure 3 shows how a single BBR flow reacts to changes of the bottleneck bandwidth in a network. Thereby, the first 55 seconds are our reproduction of [1, Fig. 3]. For equal network conditions, no significant differences are visible. The sending rate, measured RTT and inflight data closely follow the doubling in BtlBw. After the bandwidth reduction, a queue is generated, as indicated by the increased RTT estimation, and drained in the following five seconds.

Instead of an additional bandwidth reduction, we tripled RTprop at the 56 s mark. The results are surprising at first. Similar to a decrease in BtlBw, BBR cannot adapt to an increase in RTprop immediately, since the minimum filter retains an old, lower value for another 10 s. When RTprop grows, the acknowledgments for the packets take longer to arrive, which increases the inflight data until the congestion window is reached. To adapt, BBR limits its sending rate, resulting in lower samples for BtlBw. As soon as the BtlBw estimate expires, the congestion window is reduced according to the new, lower BDP. This happens repeatedly until the old minimum value for RTprop is invalidated (at approx. 62 s). Now, BBR learns about the new value and increases the sending rate again to match BtlBw with exponential growth.

While this behavior is not ideal and can cause problems, the repercussions are not severe for two reasons. First, even though the sending rate drops, the inflight data does not decrease compared to before the RTT increase. Second, it is unlikely that such a drastic change in RTT happens in the Internet in the first place.

The RTT reduction at 76 s is adapted instantly because of the RTT minimum filter.

Figure 3 also validates that our framework can sample events detailed enough ($\triangle t = 40\,\mathrm{ms}$), as both Probe Bandwidth (small spikes) and Probe RTT phases (large spikes every 10 s) are displayed accurately. However, in general we use $\triangle t = 200\,\mathrm{ms}$ for less overhead.

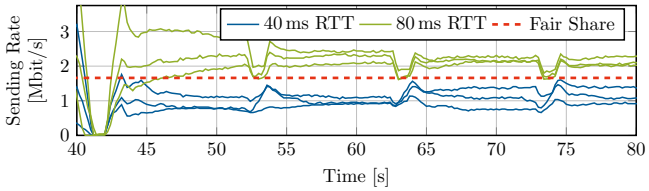


Figure 4: RTT unfairness for multiple flows with two groups of RTTs

B. RTT Unfairness

The RTT unfairness of BBR is visualized in [6, Fig. 1]. Two flows share a bottleneck of 100 Mbit/s, one flow having a larger RTT than the other (10 ms and 50 ms). The flow with shorter RTT starts three seconds before the other. We set the bandwidth to 10 Mbit/s and adapted all other parameters. Our reproduced results (not shown) only differ slightly: The larger flow receives about 10 % less of the bandwidth.

As shown in Figure 4, the behavior can also be observed when increasing the number of flows. Flows with equal RTT converge to a fair share within their group, however, groups with higher RTT claim a bigger share overall.

C. Bottleneck Overestimation for Multiple Flows

BBR overestimates the bottleneck when competing with other flows, operating at the inflight data cap [4]. The analysis

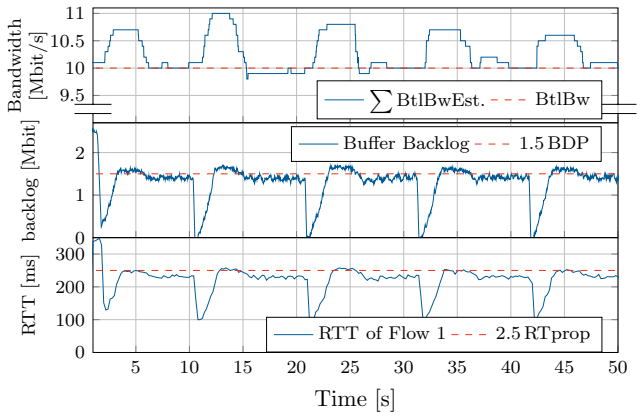


Figure 5: BDP overestimation for five flows with a 100 ms RTprop and 10 Mbit/s bottleneck (5 BDP buffer)

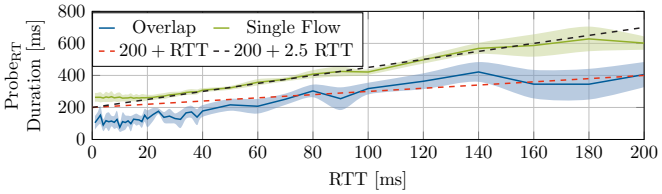


Figure 6: Overlapping Probe RTT phase duration of 5 flows

of Hock et al. predicts 2 BDP $\leq \sum_i \text{inflight}_i < 2.5$ BDP. Our experiments using a large enough buffer size of 5 BDP reproduce the results of this formal analysis as shown in Figure 5. For five simultaneously started BBR flows, the sum of the BBR estimations of BtlBw exceeds the real BtlBw after each Probe RTT phase, increasing the estimation towards the inflight cap. The backlog of the bottleneck buffer is kept at 1.5 BDP resulting in a total of 2.5 BDP.

1) Insufficient Draining of Queues During Probe RTT: To measure the correct RTprop value, all flows need to simultaneously drain the queue in Probe RTT. Figure 6, displaying the duration of the Probe RTT phase for five flows and the overlap thereof, shows that this is not the case. For RTTs below 40 ms the overlap is only half of the duration of the Probe RTT phase (200 ms + RTT). This is because all flows enter Probe RTT at slightly different times even though the flows are synchronized. As a consequence, the queue is not drained enough and BBR overestimates the bottleneck.

For high RTTs the overlap exceeds the theoretic maximum of 200 ms + RTT. Indeed, the duration of the Probe RTT phase for each individual flow equals 200 ms + 2.5 RTT. This is because when the previous RTprop value expires, triggering the Probe RTT phase, BBR chooses the newest measured RTT as RTprop [15]. As this value, however, is based on a measurement outside of the Probe RTT phase, it is influenced by the 2.5 BDP overestimation. As a consequence, the Probe RTT phase is longer, reducing the performance of BBR.

2) Retransmissions for Shallow Buffers: BBR is susceptible to shallow buffers as it overestimates the bottleneck, not recognizing that the network is strongly congested, since packet loss is not interpreted as congestion. Cardwell et al. have shown that BBR's goodput will suffer if the buffer cannot

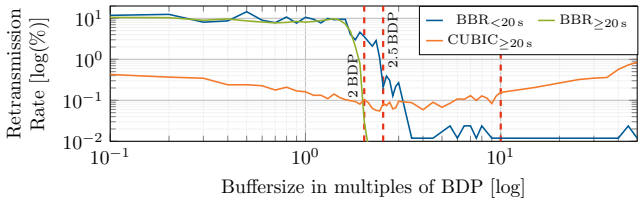


Figure 7: Retransmissions per second for 5 simultaneously started flows with different bottleneck buffer sizes

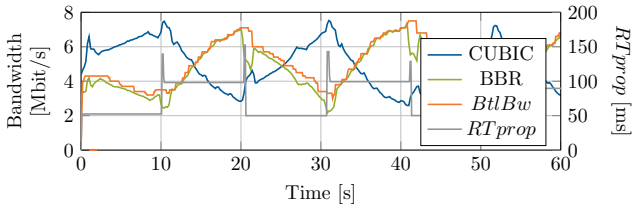


Figure 8: Competing BBR and CUBIC flow

hold the additional 1.5 BDP [5].

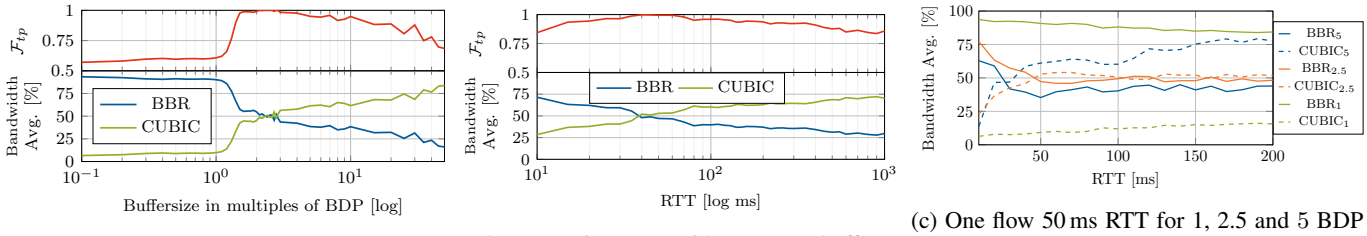
We reproduced this effect by analyzing the relation between bottleneck buffer size and caused retransmissions for both BBR and CUBIC (cf. Figure 7). Five TCP flows are started simultaneous and share a 10 Mbit/s, 50 ms bottleneck (BDP = 500 kbit). We compute the retransmission rate for different buffer sizes at the bottleneck for BBR and CUBIC individually. $\text{BBR}_{<20\,\text{s}}$ shows the first 20 s of the test including only startup and synchronization phases. $\text{BBR}_{\geq 20\,\text{s}}$ and $\text{CUBIC}_{\geq 20\,\text{s}}$ represent steady-state operation after 20 s.

For shallow buffers up to 2 BDP retransmission for BBR exceeds the amount for CUBIC by a factor of 10. This is a consequence of the buffer overestimation, in contrast to CUBIC's adaption of the congestion window for loss events. Between 2 BDP and 2.5 BDP loss only occurs during the startup and synchronization phases (before 20 s) for BBR. This is because of the initial aggressive bandwidth claiming. For even larger buffers BBR is not susceptible to loss.

CUBIC, as loss-based algorithm, produces loss with all buffer sizes during congestion avoidance phase. However, for small buffer sizes it is a factor of 10 below BBR. Only when exceeding 10 BDP = 5 Mbit a rise in retransmissions is visible for CUBIC. This is because of taildrop, increasing the repercussions of a single loss event. However, buffers with this large capacity are not realistic in the Internet [8] and therefore only pose a theoretic problem.

D. Inter-protocol Behavior With CUBIC

In the best case, a competing BBR and CUBIC flow reach an oscillating steady-state [5]. This is caused by the RTprop estimation of BBR as shown in Figure 8. CUBIC's aggressive probing for bandwidth causes the queues to fill up, resulting in BBR to measure a higher delay, increasing its BDP. In turn, this causes packet loss, resulting in reduced data inflight for CUBIC. Once the queue is drained, CUBIC starts to probe again, while BBR measures the correct RTprop value. This oscillation results in $\mathcal{F}$ being constantly low, however, both



(a) Increasing buffer with 50 ms RTT

(b) Increasing RTT with 2.3 BDP buffer

(c) One flow 50 ms RTT for 1, 2.5 and 5 BDP buffer

Figure 9: One CUBIC vs. one BBR flow for changing network conditions

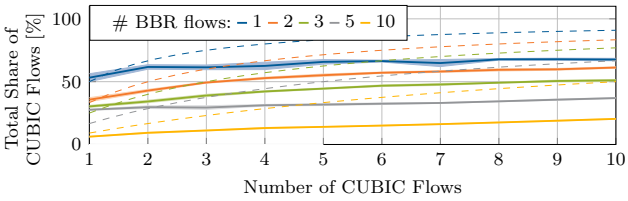


Figure 10: Bandwidth share of different number of CUBIC and BBR flows competing. Dashed lines show fair share.

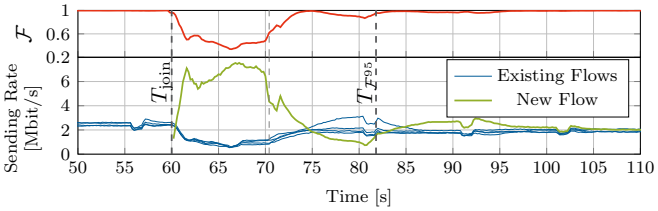


Figure 11: BBR inter-flow synchronization behavior

flows reach an equal average throughput. For the following analysis related to the inter-protocol behavior we use $\mathcal{F}_{tp}$ as fairness index based on the average throughput.

The size of the bottleneck buffer is crucial for the fairness between competing BBR and CUBIC flows [1, 4]. Figure 9a shows our reproduction of this result, displaying the bandwidth share and fairness for one BBR and one CUBIC flow for different bottleneck buffer sizes. Up to 1.5 BDP buffer size, BBR causes constant packet loss as explained in the previous section. CUBIC interprets this as congestion signal and reduces its sending rate. Up to 3 BDP both flows reach a fair share, while for further increasing buffer sizes CUBIC steadily claims more. The reason is that CUBIC fills up the ever growing buffers. For BBR this results in ever growing Probe RTT phases, i.e., reduced sending rate. The length of and the gap between Probe Bandwidth phases increases too, reducing BBR's ability to adapt. However, these buffer sizes pose only a theoretical problem (cf. Section IV-C2).

While showing the same overall behavior, RTT changes have a smaller influence on the fairness if applied to both flows as shown in Figure 9b. For all tested RTTs the fairness remained above 80 %. However, when fixating one flow at 50 ms RTT and varying the RTT of the other flow, unfairness emerges (Figure 9c). For small RTTs or shallow buffers BBR suppresses CUBIC for the already discussed reasons. In the other cases, the bandwidth share remains independent of the RTT. Only when having large buffers, CUBIC gains increasing shares with increasing RTT. Our conclusion is that the fairness between CUBIC and BBR largely depends on the botteneck buffer size, while the RTT only has a small impact.

Lastly, we evaluate how the number of flows competing with each other influences the throughput share per congestion avoidance algorithm. Figure 10 shows that CUBIC is suppressed independent of the number of flows in a scenario with 50 ms RTT and 2.5 BDP bottleneck buffer. A single BBR flow claims more bandwidth than its fair share already when competing against two CUBIC flows. In fact, independent of the number of BBR and CUBIC flows, BBR flows are always able to claim at least 35 % of the total bandwidth.

V. Inter-flow Synchronization

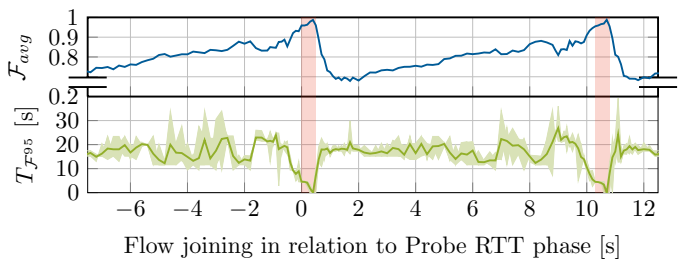
Different BBR flows synchronize themselves to avoid faulty estimations, e.g., when one flow probes for bandwidth causing a queue to form at the bottleneck, while another probes for RTT. In contrast to loss-based algorithms, this does not correlate with congestion, as the flows are impervious to loss.

A. Theory & Questions

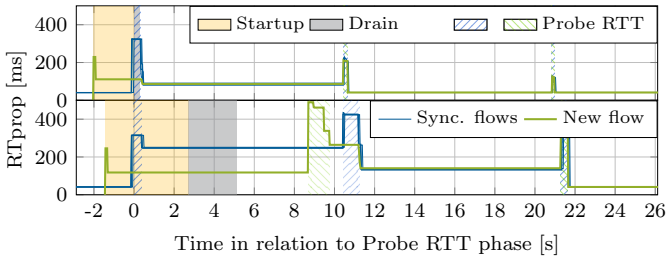
Cardwell et al. demonstrate in [1, Fig. 6] how different BBR flows synchronize whenever a large flow enters the Probe RTT phase. We visualize the process in Figure 11 with one new flow joining four already synchronized flows. The new flow immediately overestimates the bottleneck link and claims a too large share of the bandwidth. 10 s later it enters Probe RTT. The flow with bigger share drains a large portion of packets from the queue, which results in all other flows measuring a better RTprop estimate. Consequently, the flows are synchronized as the RTprop samples of all flows expire at the same time, causing them to enter Probe RTT together at the 81 s mark. Considering the fairness, it takes approximately 35 s after the new flow joined until equilibrium is reached.

To maximize performance, BBR should only spend 2% of time in Probe RTT [1, 15]. Therefore, new flows have trouble to measure the correct RTprop as active flows likely probe for more bandwidth and create queues. It causes the new flow to overestimate the BDP, inducing queuing delay or packet loss.

This raises two questions regarding the synchronization behavior of BBR flows: Is there an optimal and worst moment regarding the time until equilibrium is reached for a single flow to join a bottleneck containing already synchronized BBR flows? And secondly we want to determine if constantly



(a) Join during different times of the Probe RTT cycle. Red area marks Probe RTT phases.



(b) Correlation between Startup/Drain and Probe RTT for joining 2 s and 1.7 s before next Probe RTT phase

Figure 12: Single BBR flow joining synchronized BBR flows

adding new flows can result in extended or accumulated unfairness.

B. Synchronization Metrics

To quantify the impact of a new flow joining we use two metrics based on Jain's fairness index $\mathcal{F}$. For better comparison we define T_{join} as the point in time when the flow of interest, i.e. the last flow, has joined the network (cf. Figure 11). As first metric, we define $T_{\mathcal{F}^{95}}$ as the point after T_{join} for which $\mathcal{F}$ remains stable above 0.95, i.e. no longer than 2 s below this threshold. Second, we compute the average fairness $\mathcal{F}_{\text{avg}}$ in the interval $[T_{\text{join}}, T_{\text{join}} + 30\,\text{s}]$.

In the following we analyze the behavior of flows with equal RTTs. We assume that all effects described in the following will scale similarly as described in Section IV-B with RTT unfairness between flows.

C. Single Flow Synchronization Behavior

To analyze the basic synchronization behavior, we use the scenario of one new BBR flow joining a network with four other BBR flows already synchronized and converged to a fair share. Figure 12a shows our experimental evaluation when joining a new flow in relation to the Probe RTT phase of the synchronized flows.

As expected, a periodic behavior is revealed, with the best case for a new flow to join being during the Probe RTT phase. It synchronizes immediately as the queues are drained and the new flow can measure the optimal RTT, leading to low $T_{\mathcal{F}^{95}}$ and high $\mathcal{F}_{\text{avg}}$. The worst case is if the flow joins directly after the other flows left the Probe RTT phase. At this point, the queue is building again as the flows keep 2 BDP inflight, resulting in the new flow severely overestimating the BDP. It remains in this state until the old flows enter Probe RTT again (up to 10 s later), draining the queue and synchronizing with the new flow. This behavior of aggressively taking bandwidth from existing flows can be harmful when many short living BBR flows join, leading to starvation of long-living flows.

In general, it lasts 20 s until $T_{\mathcal{F}^{95}}$ is reached, but the later the new flow joins during the cycle, the higher varies $T_{\mathcal{F}^{95}}$ (10 to 30 s). The local optimum when joining 2 s before the Probe RTT phase with $T_{\mathcal{F}^{95}} = 10\,\text{s}$ is because the existing flows enter the Probe RTT phase while the new flow drains after the Startup as shown in Figure 12b. Consequently, all flows drain

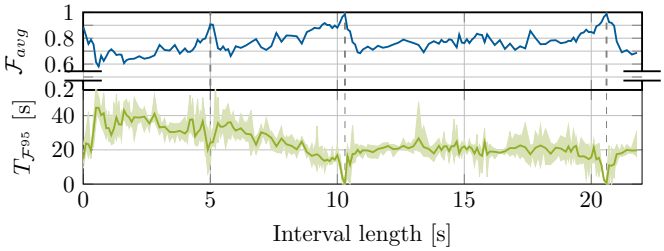


Figure 13: Different join intervals for subsequent flows

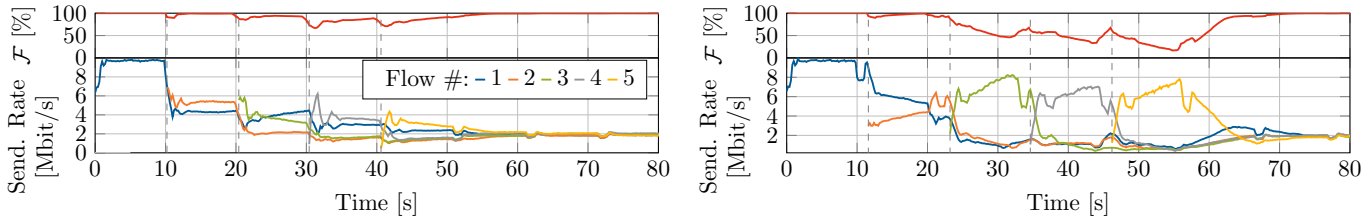
the queue and measure a new optimal RTprop, synchronizing immediately, yet overestimating the bottleneck because the queue created during Startup is not entirely drained yet. In contrast, the worse case directly afterwards (1.7 s before next Probe RTT) with $T_{\mathcal{F}^{95}} = 22\,\text{s}$ is caused by the existing flows entering Probe RTT, draining the queue, while the new flow is in Startup. This causes the new flow to drastically overestimate the bottleneck until leaving Startup, suppressing other flows.

Considering the prevalence of short-lived flows in the Internet [19, 2], this high $T_{\mathcal{F}^{95}}$ value poses a significant disadvantage of TCP BBR. Initially, flows during this time suppress other flows through unfair bandwidth claims, which is only solved when reaching a fair share.

D. Accumulating Effects

To evaluate if negative effects of multiple flows joining can accumulate, i.e. whether the duration of unfairness can be prolonged, we change the scenario to have a new flow join every x seconds up to a total of five BBR flows (cf. Figure 13).

Optima are visible for intervals matching the duration of the Probe RTT phase of the already active flows at approximately 10 s and 20 s. When all flows join at the same time, they all measure a good RTprop value within the first few packets, synchronizing them immediately. For intervals smaller than 10 s accumulating effects are visible as new flows rapidly join, not allowing the fairness to stabilize. As for a single flow, $T_{\mathcal{F}^{95}}$ and $\mathcal{F}_{\text{avg}}$ improve with increasing interval. For flows joining every 5 s an additional local optimum is visible as every second flow joins during the Probe RTT phase of the other flows. For intervals larger than one Probe RTT cycle (after flows leave Probe RTT, approximately 10.5 s), $T_{\mathcal{F}^{95}}$ and $\mathcal{F}_{avg}$ show the behavior for a single flow joining. This is because all prior



(a) 10.1 s join interval (during Probe RTT)

(b) 10.5 s join interval (immediate after Probe RTT)

Figure 14: Identified best/worst case join intervals

flows have already synchronized, resulting in them already converging towards an equilibrium before the next flow joins.

Analyzing the effect of a new flow joining on individual existing flows, e.g. the longest running flow, is difficult for the lack of a good metric. We therefore select the best and worst case join intervals displayed in Figure 14 for a visual analysis. As the minimum value of $\mathcal{F}$ depends on the number of flows $(1/n)$, it is normalized using percentages.

Figures 14a and 14b show the effects of subsequent flows joining during (best case) or immediately after (worst case) the Probe RTT phase. Similar to the effects on the last flow joining, existing flows are only influenced by the timing of the next flow joining. Within the group of synchronized flows, they converge to their fair share. The synchronization itself depends on the timing and happens at most after 10 s. The resulting unfairness is only caused by the new flow. The overall time until bandwidth equilibrium is approximately 55 s and 70 s, respectively. We attribute the 15 s difference to the longer synchronization phase in the latter case (10 s) and bigger unfairness thereof.

Throughout all our tests we encountered rare cases where $T_{\mathcal{F}^{95}}$ extended for up to 50 s, which are not reproducible. We attribute this instability to the sensibility of the join timing.

Summarizing, the fair sharing of bandwidth is intertwined with the timing of new flows joining the network. Except during the brief Probe RTT phase, equilibrium is only reached after 20 s and can extend up to 30 s. However, there are no effects accumulating beyond the interval of one Probe RTT phase. The timing only has a short term effect on the amplitude of unfairness, not $T_{\mathcal{F}^{95}}$.

VI. Related Work

Different definitions of and processes to reach reproducibility exist [20], e.g. as a three stage process as defined by an ACM policy [21]. Thereby, the minimum level is **repeatability**. It refers to recreating the results for an experiment conducted by the same scientists with the same tools. The term **replicability** is used for results that can be reproduced by other scientists given the same experiment setting. Finally, **reproducibility** defines that results can be validated in different experiments, by different scientists and tools.

Our paper contributes to these quality aspects by reproducing results of other scientists with different methods (i.e. *reproducibility*). By providing our framework as open source software, we increase the value of our results by allowing others to replicate them (i.e. *replicability*).

A. Reproducible Measurements with Network Emulation

Handigol et al. [22] have shown that various network performance studies could be reproduced using Mininet. The Mininet authors published an editorial note [23] in 2017, wherein they describe efforts in reproducing research. They reproduced performance measurements of DCTCP, Multi-Path TCP (MPTCP), the TCP Opt-ack Attack, TCP Fast Open, and many more. Other research groups used Mininet in studies about TCP, such as the work from Paasch et al. [24], with a performance evaluation of MPTCP. Girardeau and Steele use Mininet in Google Cloud VMs to perform simple BBR measurements [25]. They use a patched kernel and, compared to our approach, their setup and runtime for one experiment is significantly higher with up to 50 minutes.

BBR support is announced to be available for the network simulator ns3 [26]. The Pantheon allows researchers to test congestion control algorithms in different network scenarios [27]. The results of Internet measurements are used to tune the parameters of emulated network paths which provides better reproducibility.

B. TCP BBR in Other Domains

BBR deployed in domains with different requirements yields varying results. Kuhn has shown promising results over SATCOM links, which have latencies in the range of 500 ms [28]. They state that a “late-comer unfairness” [28] exists. Leong et al. claim that BBR can be further improved for mobile cellular networks [29], which is a recent research area of Cardwell et al. [5]. Kakhki et al. integrated BBR for QUIC, however, state that it is not yet performing well [30].

VII. Conclusion

We presented a framework for TCP congestion control measurements focusing on flexibility, portability, reproducibility and automation. We used Mininet to emulate different user-configured flows. Experiments run without user interaction and produce a report containing graphs visualizing 14 metrics. We reproduced related work to validate the applicability of our approach using emulation.

Furthermore, we summarized the state of the art for analysis for TCP BBR and extend existing insights in several aspects. In particular, we have shown that the algorithm to determine the duration of the Probe RTT phase is flawed and that in most cases BBR and CUBIC do not share bandwidth in a fair manner. Our final contribution is an experimental

analysis of the synchronization mechanism. We identified two primary problems. Depending on the timing of new flows joining existing flows in relation to their Probe RTT phase, bandwidth can be shared severely unfair. This boils down to BBR's general problem of overestimating the BDP. The second problem is the time until a bandwidth equilibrium is regained. This can last up to 30 s, which is bad for short-lived flows, common in today's Internet. We identified that this is correlated with the trigger for synchronization, i.e. the Probe RTT phase, draining the queues. Consequently, without reducing the time between Probe RTT phases, the worst case time until flows synchronize cannot be improved further.

Our framework as well as the raw data for all figures presented is available online [7] for replicability of our results and to allow further investigations by the research community.

Acknowledgment

This work was supported by the High-Performance Center for Secure Networked Systems and the German BMBF project SENDATE-PLANETS (16KIS0472).

References

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based Congestion Control," *ACM Queue*, vol. 14, no. 5, 2016.

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, 2008.

[3] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," in *Proceedings of the International Conference on Communications*, vol. 43, 1979.

[4] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *25th IEEE International Conference on Network Protocols (ICNP 2017)*, 2017.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, I. Swett, J. Iyengar, and V. Vasiliev, "BBR Congestion Control: IETF 100 Update: BBR in shallow buffers," *IETF 100*, 2017, Presentation Slides. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers/

[6] S. Ma, J. Jiang, W. Wang, and B. Li, "Towards RTT Fairness of Congestion-Based Congestion Control," *CoRR*, vol. abs/1706.09115, 2017. [Online]. Available: http://arxiv.org/abs/1706.09115

[7] Framework and Data Publication. [Online]. Available: https://gitlab.lrz.de/tcp-bbr

[8] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Commun. ACM*, vol. 55, no. 1, 2012.

[9] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," Tech. Rep., 2009.

[10] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, 1995.

[11] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats *et al.*, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015.

[12] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks*, 2017.

[13] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for high-speed and long Distance Networks," in *Proceedings-IEEE INFOCOM*, 2006.

[14] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the 2011 ACM SIGCOMM Conference*, vol. 41, no. 4. ACM, 2011.

[15] N. Cardwell, Y. Cheng, S. Yeganeh, and V. Jacobson, "BBR Congestion Control," Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, 2017. [Online]. Available: http://www.ietf.org/internet-drafts/draft-cardwell-iccrg-bbr-congestion-control-00.txt

[16] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[17] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," Internet Requests for Comments, RFC Editor, RFC 5166, 2008.

[18] R. Jain, D.-M. Chiu, and W. R. Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984, vol. 38.

[19] S. Ebrahimi-Taghizadeh, A. Helmy, and S. Gupta, "TCP vs. TCP: a systematic study of adverse impact of short-lived TCP flows on long-lived TCP flows," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2. IEEE, 2005.

[20] D. G. Feitelson, "From Repeatability to Reproducibility and Corroboration," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, 2015.

[21] O. Bonaventure, "April 2016: Editor's message," in *ACM SIGCOMM CCR*, 2016.

[22] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-based Emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. ACM, 2012.

[23] L. Yan and N. McKeown, "Learning Networking by Reproducing Research Results," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 2, 2017.

[24] C. Paasch, R. Khalili, and O. Bonaventure, "On the Benefits of Applying Experimental Design to Improve Multipath TCP," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. ACM, 2013.

[25] B. Girardeau and S. Steele. Reproducing Network Research (CS 244 '17): Congestion-based Congestion Control With BBR. [Online]. Available: https://reproducingnetworkresearch.wordpress.com/2017/06/05/cs-244-17-congestion-based-congestion-control-with-bbr/

[26] C. A. Grazia, N. Patriciello, M. Klapez, and M. Casoni, "A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control?" in *Proceedings of the 14th International Conference on Telecommunications (ConTEL)*. IEEE, 2017.

[27] Pantheon: The Training Ground for Internet Congestion Control Research. [Online]. Available: http://pantheon.stanford.edu

[28] N. Kuhn, "MPTCP and BBR performance over Internet satellite paths," *IETF 100*, 2017, Presentation Slides. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-mptcp-and-bbr-performance-over-satcom-links/

[29] W. K. Leong, Z. Wang, and B. Leong, "TCP Congestion Control Beyond Bandwidth-Delay Product for Mobile Cellular Networks," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017.

[30] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a Long Look at QUIC," in *Proceedings of the 2017 Internet Measurement Conference*, 2017.

Cellular Controlled Delay TCP (C2TCP)

Soheil Abbasloo, Tong Li, Yang Xu, H. Jonathan Chao
New York University
Email: {ab.soheil, tl1914, yang, chao}@nyu.edu

***Abstract*—Cellular networks have special characteristics including highly variable channels, fast fluctuating capacities, deep per user buffers, self-inflicted queuing delays, radio uplink/downlink scheduling delays, etc. These distinguishing properties make the problem of achieving low latency and high throughput in cellular networks more challenging than in wired networks. That's why in this environment, TCP and its flavors, which are generally designed for wired networks, perform poorly.**

To cope with these challenges, we present C2TCP, a flexible end-to-end solution targeting interactive applications requiring high throughput and low delay in cellular networks. C2TCP stands on top of loss-based TCP and brings it delay sensitivity without requiring any network state profiling, channel prediction, or complicated rate adjustment mechanisms. The key idea behind C2TCP is to absorb dynamics of unpredictable cellular channels by investigating local minimum delay of packets in a moving time window and react to the cellular network's capacity changes very fast.

Through extensive trace-based evaluations using traces from five commercial LTE and 3G networks, we have compared performance of C2TCP with various TCP variants, and state-of-the-art schemes including BBR, Verus, and Sprout. Results show that on average, C2TCP outperforms these schemes and achieves lower average and 95th percentile delay for packets.

I. Introduction

Cumulative data traffic growth in cellular networks has increased more than 1200% over recent five-year period and in the first quarter of 2017, total cellular internet traffic reached to nearly 9500 PetaByte per month globally [1]. This growing mode of internet access on the one hand has provided opportunities for cellular network carriers with more demand for new applications like augmented reality, virtual reality, online gaming, real time video streaming, and real time remote health monitoring. On the other hand it has brought new challenges for cellular carriers due to ultra low latency and high throughput requirements of those interactive applications.

Cellular networks differ noticeably from their wired counter parts. They experience highly variable channels, fast fluctuating capacities, self-inflicted queuing delays, stochastic packet losses, and radio uplink/downlink scheduling delays. These differences make the problem of achieving low latency and high throughput in cellular networks more challenging than in wired networks. TCP and its variants which are the main congestion control mechanisms to control the delay and throughput of flows are known to perform poorly in cellular networks [2]–[6].

In this paper we present *C2TCP*, a ***C**ellular **C**ontrolled delay **TCP*** to address mentioned challenges in cellular networks for achieving low delay and high throughput. Our main philosophy is that achieving good performance does not necessarily comes from complex rate calculation algorithms or complicated channel modelings.[1] The key idea behind C2TCP's design is to absorb dynamics of unpredictable cellular channels by investigating local minimum of packets' delay in a moving time window. Doing that, C2TCP stands on top of a loss-based TCP such as Cubic [8] and brings it delay sensitivity. There is no network state profiling, channel prediction, or any complicated rate adjustments mechanisms involved.

There is always a trade-off between achieving lowest delay and getting highest throughput. J. Jaffe in [9] has proved that no distributed algorithm can converge to the operation point in which both the minimum RTT and maximum throughput are achieved. Considering that trade-off, C2TCP provides a flexible end-to-end solution which allows applications to choose their level of delay sensitiveness, even after the start of their connection.

We have implemented C2TCP in Linux Kernel 4.13, on top of Cubic, conducted extensive trace-driven evaluations (detailed in section IV) using data collected in prior work ([10] and [2]) from 5 different commercial cellular networks in Boston (T-Mobile's LTE and 3G UMTS, AT&T's LTE, and Verizon's LTE and 3G 1xEV-DO) in both directions, and compared performance of C2TCP with several TCP variants (including Cubic [8], NewReno [11], and Vegas [12]) and different state-of-the-art end-to-end schemes including BBR [13], Sprout [2], and Verus [4]. Our results show that C2TCP outperforms these end-to-end schemes and achieves lower average and 95th percentile delays for packets. In particular, on average, Sprout, Verus, and BBR have $3.41\times$, $10.36\times$, and $1.87\times$ higher average delays and $1.44\times$, $27.36\times$, and $2.06\times$ higher 95th percentile delays compared to C2TCP, respectively. This great delay performance comes at little cost in throughput. For instance compared to Verus (which achieves the highest throughput among those 3 state-of-the-art schemes), C2TCP's throughput is only $0.22\times$ less.

Also, in section IV-B, we compared our end-to-end solution, C2TCP, with CoDel [14], an AQM scheme that requires modification on carriers network, and show that C2TCP can outperform CoDel too. Moreover, we examined fairness of C2TCP, compared it with several other algorithms, and showed that it provides good fairness properties. Finally, we investigated the loss resiliency of C2TCP in case of stochastic packet

[1]It is already a known fact that predicting cellular channels is hard [4], [7]

ISBN 978-3-903176-08-9 2018 IFIP

losses unrelated to congestion in cellular networks. Among algorithms that we examined only C2TCP, BBR, and Vegas show good resiliency in high rates of packet losses.

II. Motivations and Design Decisions

Flexible End-to-End Approach: One of the key distinguishing features of cellular networks is that cellualr carriers generally provision deep per user queues in both uplink and downlink directions at base station (BS). This leads to issues such as self-inflicted queuing delay [2] and bufferbloat [6], [15]. A traditional solution for these issues is using AQM schemes like RED [16]; however, correct parameter tuning of these algorithms to meet requirements of different applications is challenging and difficult. Although newer AQM algorithms such as CoDel [14] can solve the tuning issue, they design queues from scratch, so deploying them in network comes with huge CAPEX cost. In addition, in-network schemes lack flexibility. They are based on "one-setting-for-all-applications" concept and won't consider that different type of applications might have different delay and throughput requirements. Moreover, with new trends and architectures such as mobile content delivery network (MCDN) and mobile edge computing (MEC) [17], content is being pushed close to the end-users. So, from the latency point of view, problem of potential large control feedback delay of end-to-end solutions diminishes if not disappears. Motivated by these shortcomings and new trends, we seek a "*flexible end-to-end*" solution without tuning difficulties.

Simplicity: Cellular channels often experience fast fluctuations and widely variable capacity changes over both short and long timescales [4]. This property along with several complex lower layer state machine transitions [5], complicated interactions between user equipment (UE) and BS [18], and scheduling algorithms used in BS to allocate resources for users through time which are generally unknown for end-users make cellular channels hard to be predictable if not *unpredictable* [4], [7]. These complexities and unpredictability nature of channels motivates us to avoid using any channel modeling/prediction and to prevent adding more complicity to cellular networks. We believe that performance doesn't always come from complexity. Therefore, we seek "*simple yet powerful*" approaches to tackle congestion issue in cellular networks.

Network as a Black-Box: In cellular networks, source of delay is vague. End-to-end delay could be due to either self-inflicted queuing delays in BS, delays caused by BS' scheduling decisions in both directions, or downlink/uplink channel fluctuations. Although providing feedback from network to users can guide them to detect the main source of delay, any new design based on requiring new feedback from network needs modifications on cellular networks. However, this comes at the CAPEX cost for cellular carriers. Therefore, we will look at cellular network as a "*black-box*" which doesn't provide us any information about itself directly.

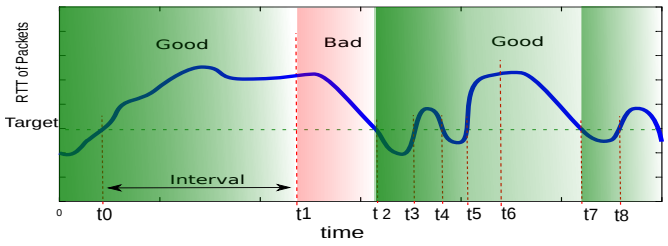


Fig. 1. The Good and the bad conditions

III. C2TCP's Algorithm

A. The Good and The Bad Conditions

Inspired by CoDel [14] and Vegas [12] designs, we define that network is in "*good-condition*" when minimum RTT observed in a monitoring time duration (called *Interval*) remains below a *Target* delay. If RTT of a packet goes above the Target and if RTTs of next packets never fall below the Target in the next *Interval*, we say network is in a *bad-condition*. So, having any RTT less than *Target* shows a good-condition at least for the next *Interval*, while instantaneous RTTs bigger than *Target* doesn't necessarily show a "*bad-condition*". Intuitively, this definition comes from the fact that one of the normal responsibilities of queues in the network is to absorb burst of traffic, so it is normal to have some temporary increase in RTT of packets. However, when more packets experience large RTTs, most likely there is something wrong in the network. Hence, history of delay should be considered as important as the current delay.

For instance, consider Fig. 1 which shows sample RTTs of packets through time. At t_0 RTT of a packet goes beyond the Target value and till $t_1 = t_0 + Interval$ no packet experiences RTT less than Target. So, at t_1 a bad-condition is detected. This bad-condition continues till t_2 when RTT goes below Target value indicating detection of a good-condition. At t_3 RTT goes above Target but since RTT becomes less than Target at t_4 ($< Interval + t_3$), we still remain in good-condition. Likewise we remain in good-condition for the next time slots.

Note that the delay responses of packets in $[t_1, t_2]$ and $[t_6, t_7]$ periods are identical. However, since the history of delay is different at t_1 and t_6, those two periods have been identified differently (first one is in a bad-condition, while the second one is in a good-condition). This example shows how we can use our simple definition to qualitatively get a sense of history without recording history of delay.

B. Main Logic

As Algorithm 1 shows, C2TCP's main logic will be triggered each time a new acknowledgment is received at the source. After detecting a bad-condition, the main question is that *if we had an in-network AQM algorithm able to detect the bad-condition, what would have it done to inform a loss-based TCP?* The answer is that it would have simply dropped the packet and caused TCP to do a harsh back-off by setting congestion window to one. So, the key idea of

Algorithm 1: C2TCP's Main Algorithm at Sender

```
1  Function pkts_acked() // process a new received Ack
2      ... /* default loss-based TCP code block */
3      rtt ⟵ current_rtt
4      now ⟵ current_time
5      if rtt < Target then
           /* good-condition */
6          Cwnd += Target/rtt
7          first_time ⟵ true
8          num_backoffs ⟵ 1
9      else if first_time then
           /* waiting phase */
10         next_time ⟵ now + Interval
11         first_time ⟵ false
12     else if now > next_time then
           /* bad-condition */
13         next_time ⟵ now + Interval/√num_backoffs
14         num_backoffs + +
           /* setting ssthresh using default TCP
           function which normally recalculates it
           in congestion avoidance phase */
15         ssthresh ⟵ recalc_ssthresh()
16         Cwnd ⟵ 1
```

C2TCP is to emulate such an impact without requiring that imaginary AQM scheme in the network. When bad-condition is detected at source, C2TCP overwrites the decision of TCP and forces congestion window to be reset to one. Each time a bad-condition is detected, the next monitoring time interval is decreased in proportion to $\frac{1}{\sqrt{n}}$ where n is number of consecutive back-offs since detecting the current bad-condition using the well-known relationship of drop rate to throughput to get a linear change in throughput [19], [20].

On the other hand, RTTs smaller than Target show room for applications to increase their throughput at cost of increase in their delay. Therefore, we break the good-condition into two phases: 1- When RTTs are smaller than Target and 2- When RTTs are larger than Target (waiting phase). In waiting phase (lines 9-12 in Algorithm 1) , C2TCP doesn't change the congestion window calculated by the loss-based TCP, which is used as base for C2TCP, and waits for transition to either bad-condition or another first phase of the good-condition. However, in the first phase, C2TCP increases the congestion window additively using equation 1 (lines 5-9 in Algorithm 1). This increase is in addition to the increase that the loss-based TCP normally does. The choice of this additive increase is to follow the well-known AIMD (Additive Increase Multiplicative Decrease) property which ensures that C2TCP's algorithm still achieves fairness among connections [21]. We have examined C2TCP's fairness in more detail in section IV-C.

$$Cwnd_{new} = Cwnd_{current} + \frac{Target}{rtt_{current}} \quad (1)$$

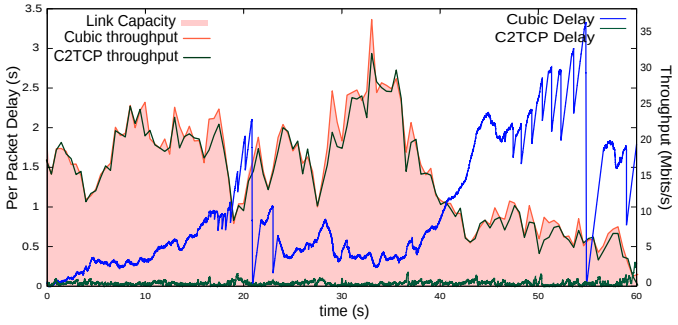


Fig. 2. Delay Response of Cubic and C2TCP

C. Why It Works

To show the improvements achieved by C2TCP and discuss the reasons, we compare the performance of C2TCP implemented on top of Cubic with Cubic following instructions described in section IV. Fig. 2 shows 60 seconds of varying capacity of a cellular link (Verizon LTE network in downlink direction measured in Boston by prior work [2]) and delay/throughput performance of C2TCP and Cubic.

1) Avoiding excessive packet sending: Due to variations in link capacity and deep per user buffers, Cubic's delay performance is poor, specially when there is a sudden drop in capacity of link after experiencing good capacity (for instance, look at $[15s - 20s]$ and $[35s - 45s]$ time periods in Fig. 2). However, C2TCP always perform very well regardless of the fast link fluctuations. The key reason is that, C2TCP always keeps *proper* amount of packets in the queues so that on the one hand, it avoids queue buildup and increase in the packet delay and on the other hand, it achieves high utilization of the cellular access link when either channel quality becomes good or BS' scheduling algorithm allows serving packets of the corresponding UE.

2) Absorbing dynamics of channel: In contrast with designs like Vegas [12] which use the overall minimum RTT of a connection during its life time, we use a moving minimum RTT. Monitoring minimum RTT in a moving time window allows us to absorb dynamics of cellular link's capacity, scheduling delays, and in general, different sources of delay in network, without need for having knowledge about the exact sources of those delays, which in practice, are hard to be known at end-hosts.

3) Cellular link as the bottleneck: Based on high demand of cellular-phone users to access different type of contents, new trends and architectures such as MEC [17], MCDN (e.g. [22]), etc. have been proposed and used recently to push the content close to the end-users. So, cellular access link known as the *last-mile* becomes the bottleneck even more than before. This insight helps C2TCP's design to concentrate on the delay performance of the last-mile and boost it.

4) Isolation of per user queues in cellular networks: Since C2TCP targets cellular networks, it benefits from their characteristics. One of the important characteristics of cellular networks is that usually different UEs get their own isolated

deep queues at BS and there is rare competition for accessing queue of one UE by flows of other UEs [2], [4], [6]. Mentioned architecture puts BS' scheduler in charge of fairness among UEs using different algorithms such as weighted round robin, or proportional fairness. This fact helps C2TCP to focus more on the delay performance and leave the problem of maintaining fairness among UEs on the last-mile to the scheduler. In addition, it is a reasonable assumption that for cellular end-users there is usually one critical flow using UE's isolated queue at BS when users are running delay sensitive applications such as virtual reality, real time gaming, real time streaming, real time video conferencing, etc. on their smartphones. C2TCP benefits from this fact too. [2]

5) What if C2TCP shares a queue with other flows: Although the main delay bottleneck in cellular network is the last-mile, there still might be concern about the congestion before the access link (for instance, in the carrier's network). The good news is that in contrast with large queues used at BS, normal switches and routers use small queues [23]. So, using well-known AIMD property ensures that the C2TCP will achieve fairness across connections [21] before the flow reaches its isolated deep buffer at BS. In section IV-C, we show good fairness property of C2TCP in the presence of other flows in such condition.

6) Letting loss-based TCP do the calculations: Another helpful insight behind C2TCP is that in contrast with delay-based TCPs, C2TCP does not directly involve packets' delay to calculate the congestion window, but let loss-based TCP, which is basically designed to achieve high throughput [8], [11], [24], [25], do most of the job. So, instead of reacting directly to every large RTT, definition of "bad-condition" helps C2TCP detect persistent delay problems in a time window and react only to them. Therefore, events impacting only a few packets (such as stochastic losses unrelated to congestion) won't impact the algorithm that much. Good resiliency of C2TCP to stochastic packet losses is investigated in section IV-D.

D. Does C2TCP work in other networks?

Our design rests on underlying architectures of cellular networks including presence of deep per user buffers at BS, exploiting an scheduler at BS which brings fairness among various UEs at the bottleneck link (last-mile), and low end-to-end control feedback delay (thanks to current technologies and trends such as MEC, MCDN, M-CORD [26], etc.). Therefore, lack of these structures will impact C2TCP's performance. For instance, for networks with very large intrinsic RTTs, end-hosts absorb the network's condition with a large delay due to the large feedback delay. So, because of that large feedback delay, C2TCP (and any other end-to-end approaches) couldn't catch fast link fluctuations and respond to them very fast.

IV. Evaluation

In this section, we evaluate performance of C2TCP using extensive trace-driven emulation and compare its performance with existing protocols under a reproducible network condition (source code is available to the community at: https://github.com/soheil-ab/c2tcp). As our trace-driven emulator, we use Mahimahi [10] and use use Iperf application to generate traffic.

Fig. 3. Topology used for evaluations

Cellular Traces: Evaluations are conducted using data collected in prior work ([10] and [2]) from 5 different commercial cellular networks in Boston (T-Mobile's LTE and 3G UMTS, AT&T's LTE, and Verizon's LTE and 3G 1xEV-DO) in both downlink and uplink directions.

Schemes Compared: We have implemented C2TCP in Linux Kernel 4.13, on top of Cubic as the loss-based TCP, though any other loss-based TCP variants can be simply used as the base. We use this implementation to compare C2TCP with the state-of-the-art end-to-end schemes including BBR [13], Verus [4], Sprout [2], and different TCP flavors including Cubic [8], Vegas [12], and NewReno [11][3]. We also compare C2TCP with CoDel [14] an in-network solution which requires to be implemented in the carrier's cellular equipment for downlink queue and in baseband modem or radio interface driver of phones for uplink queue. To do that, we use Cubic at server/client sides and use CoDel scheme as queue management scheme in the network. For C2TCP, unless it is mentioned, we set both *Target* and *Interval* to 100ms. We examine sensitivity of C2TCP to these two parameters in section IV-E.

Metrics: We use 3 main performance metrics for evaluations: average throughput (in short, throughput), average per packet delay (in short, delay), and 95th percentile per packet delay (in short, 95th percentile delay). Average throughput is the total number of bits received at the receiver divided by the experiment's duration. Per packet delay is end-to-end delay which is experienced by a packet from the time being sent to the time being received excluding the propagation delay. Moreover, we investigate the fairness of different schemes. Fairness criterion shows the behaviour of different schemes when there is(are) another normal TCP flow(s) in network. In addition to these metrics, we compare resiliency of different schemes to stochastic packet losses (unrelated to congestion) which might occur in cellular networks.

Topology: We mainly use 3 entities (equipped with Linux OS) shown in Fig. 3 for these evaluations. The first one represents the server, the 2nd one emulates the cellular access channel (and BS) using Mahimahi toolkit, and the 3rd one represents the UE. The RTT is around 40ms.

[2]If not, users can simply prioritize their flows locally, and send/request the highest priority one first.

[3]We saw a bug in LEDBAT's implementation [27] which has been confirmed in our conversations with its authors, so we didn't include the result of its performance here

A. Comparison with End-to-End Schemes

Fig. 4 shows the performance of various end-to-end schemes in our extensive trace-driven evaluations for 5 different measured networks. For each network, there are 2 graphs representing 2 data transfer directions (uplink and downlink), and for each direction there are 2 charts, one showing the average delay and throughput, and the other one illustrating 95th percentile delay and throughput. Schemes achieving higher throughput and lower delay (up and to right region of graphs) are more desirable.

The overall results averaged across all evaluations have been shown in Table I. C2TCP achieves the lowest average delay and the lowest 95th percentile delay among all schemes, while compromising throughput slightly. For instance, on average, compared to Cubic, C2TCP decreases the average delay more than $200\times$, while compared to Cubic which achieves the highest throughput, it only compromises throughput $0.27\times$.

TABLE I
OVERALL RESULTS AVERAGED ACROSS ALL TRACED NETWORKS

	Thr.(Mbps)	Avg. Delay(ms)	95th%tile Delay(ms)
C2TCP	4.235	54.1	127.2
NewReno	5.768	7688.3	16934.5
Vegas	3.259	60.4	199.1
Cubic	5.772	11015	23630.2
Sprout	3.369	185	183.4
Verus	5.408	560.7	3481.4
BBR	4.796	101.3	262.4

Generally, results for different traces in Fig. 4 show a common pattern. As expected, Cubic and NewReno achieve the highest throughput among different schemes. The reason is that since they are not sensitive to delay, they simply buildup queues. Therefore, they will achieve higher utilization of the cellular access link when channel experiences good quality. Vegas and Sprout can achieve low delays but they compromise the throughput. Verus performs better than schemes such as Cubic and NewReno and achieves lower average and 95th percentile delays. However, its delay performance is far from the delay performance of Sprout, BBR, Vegas, and C2TCP for almost all traces. Design of BBR is based on first getting good throughput and then reaching good delays [13]. This explains why BBR can get good throughput while its delay performance is not good. The main idea behind Sprout is to predict the future cellular link's capacity and send packets to the network cautiously to achieve low 95th percentile delay for packets. We observed that Sprout can achieve good delay performance, but it sacrifices throughput. C2TCP tries to achieve low per packet delay while having high throughput. Results confirm that C2TCP achieves the low delay across each of 10 links while maintaining a good throughput performance.[4]

B. Comparison with an In-Network Scheme

Now, we compare performance of our end-to-end solution C2TCP with CoDel, an in-network solution which is one of the schemes that inspired us. To do that, we add CoDel AQM algorithm to both uplink and downlink queues in Mahimahi and use Cubic at the end hosts. Tabe II shows the overall results averaged across all traced networks. Using CoDel improves the delay performance of Cubic while degrading its throughput. It is worth mentioning that to have in-network solutions such as CoDel, cellular carriers should install these in-network schemes inside their base stations and in base band modem or radio-interface drivers on cellular phones, while an end-to-end scheme like C2TCP only requires updated software at cellular phones, and thus is much easier to be deployed. We show in section IV-E that by changing *target* parameter of C2TCP we can get delay performances better than CoDel's delay performances at the cost of trading throughput.

TABLE II
OVERALL RESULTS AVERAGED ACROSS ALL TRACED NETWORKS

	Thr.(Mbps)	Avg. Delay(ms)	95th%tile Delay(ms)
C2TCP	4.235	54.1	127.2
CoDel+Cubic	4.001	39	94.8

C. Fairness

Here, we examine the fairness property of C2TCP. Here, fairness property means that in the presence of other TCP flows, how much fair the bandwidth will be shared among the competing flows. Usually, a scheme that is too aggressive is not a good candidate since it may starve flows of other TCP variants. To evaluate the fairness, we send one Cubic flow from one server to an UE. Choosing Cubic as the reference TCP rests on the fact that Cubic is the default TCP in Linux and Android OS which takes more than 60% of smart phone/tablet market share [28]. Then, after 30 seconds, we start sending another flow using the scheme under investigation from another server to the same UE and will report the average throughput gained by both flows through time. When there are unlimited queues in BS, there will be no scheme which can get a fair portion of bandwidth when the queue is already being filled by another aggressive flow [2]. So, to have a fair comparison, as a rule of thumb, we set queue size to the BDP (bandwidth delay product) of the network. Here, the access link's bandwidth and RTT are 24Mbps and 40ms respectively.[5]

Fig. 5 shows the results for different schemes. The results indicate that BBR and Verus are so aggressive and will get nearly all the bandwidth from Cubic flow, while Vegas' share of link's bandwidth cannot grow in the presence of Cubic.

The main idea of BBR is to set congestion window to the BDP (bandwidth delay product) of the network. To do that, it measures min RTT and delivery rate of the packets. When queue size is at the order of BDP, BBR fully utilizes the queue and will not reserve room for the other flows. Therefore, in our case, cubic flow experiences extensive packet

[4]It is worth mentioning that all experiments have been repeated several times to make sure that the results presented here are not impacted by the random variations.

[5]Sprout's [2] main design idea is to forecast the cellular access link capacity using a varying Poisson process, so this scheme won't work properly when link bandwidth is constant. Therefore, to have fair comparison, we don't include performance results of this scheme here.

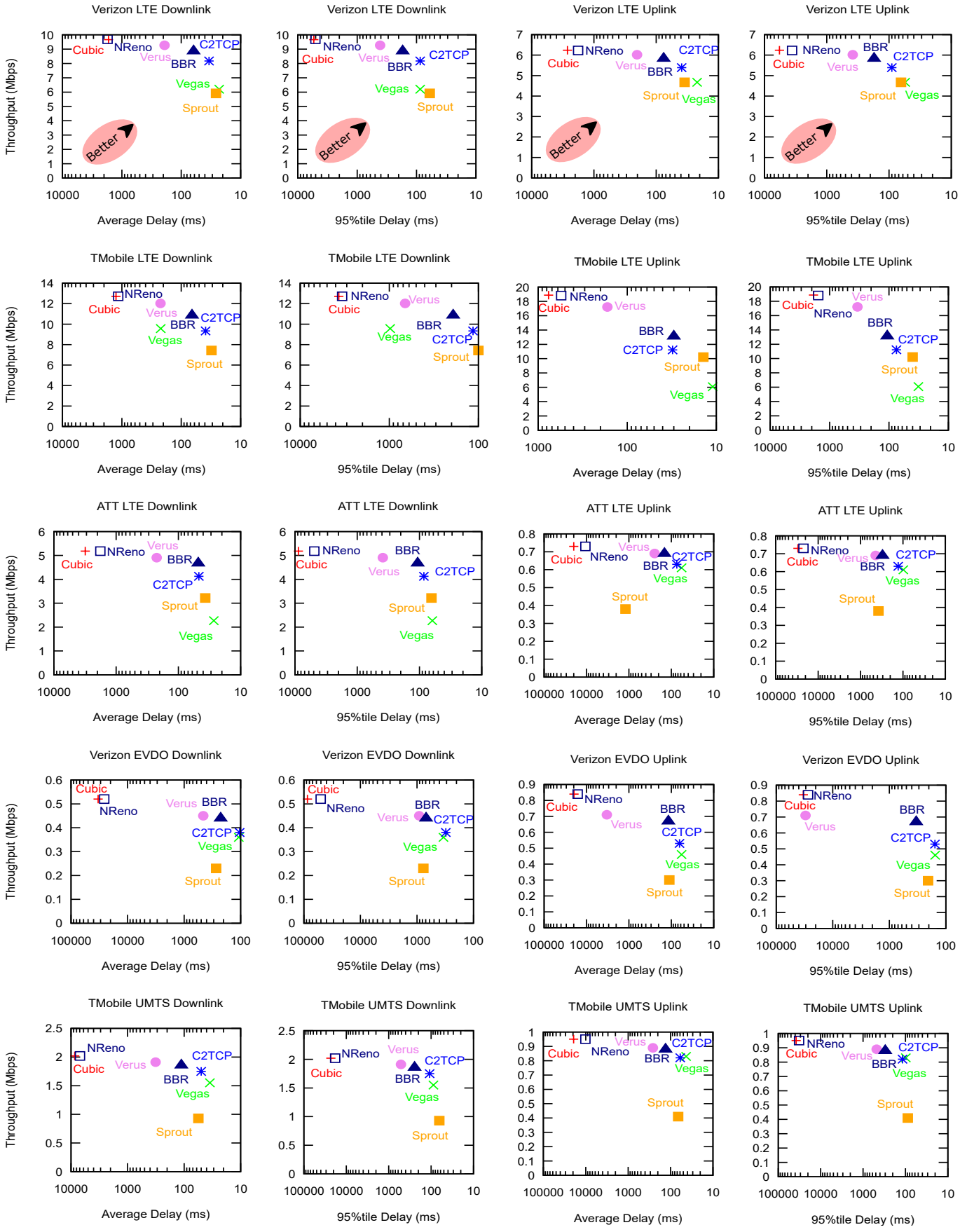


Fig. 4. Throughput, average delay, and 95th percentile delay of each scheme over traced cellular links (x axis is in logorithmic scale)

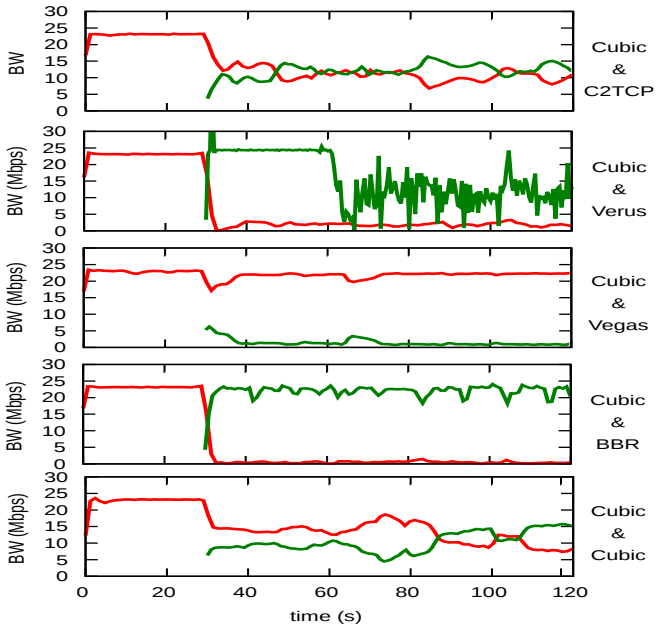


Fig. 5. Share of bandwidth among Cubic, started at time 0, and other schemes, started at $30s$ (schemes tested are named on right side of each graph)

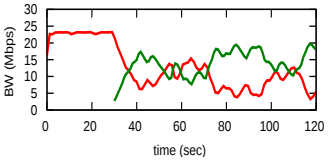


Fig. 6. Share of bandwidth among NewReno, started at 0, and C2TCP

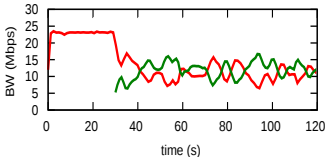


Fig. 7. Share of bandwidth among two C2TCP flows

drops and won't achieve its fair share of the bandwidth. Vegas changes its congestion window based on the minimum and the current RTT of the packets. Presence of cubic flow's packets in the queue impact both minimum RTT and current RTT measurements of Vegas. That's why Vegas cannot increase its throughput and get its share of the bandwidth from cubic flow.

In both cases, either being very aggressive or having no aggressiveness, the fairness characteristic of these schemes is not desirable. However, as Fig. 5 illustrates, C2TCP can share the bandwidth with Cubic flow fairly. In our evaluations, C2TCP is implemented over Cubic. To show that C2TCP's fairness property is not because the competing flow in test is Cubic, we replace Cubic flow with a NewReno flow and do the test again. Fig. 6 shows the result indicating the same fairness property of C2TCP.

Also, to examine the fairness criterion of the C2TCP in the presence of another C2TCP flow, we use the previous setup and replace the Cubic flow with a C2TCP flow. Results shown in Fig. 7 declare that C2TCP is fair to other C2TCP flow in the network. That being mentioned, C2TCP is friendly to other TCP flows and can achieve good fairness property.

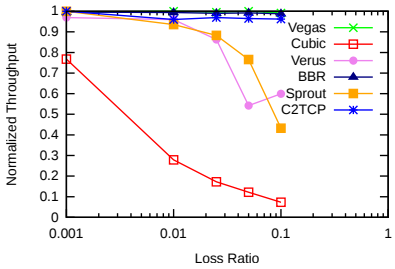


Fig. 8. Resiliency of schemes to packet losses not caused by congestion

D. Loss Resiliency

Although in cellular networks, different techniques such as HARQ [18] have been used to reduce the impact of stocastic packet losses in access link (which are not caused by congestion), there still could be stocastic packet losses in practice. So, in this section, we investigate the resiliency of different schemes to packet loss not casued by congestion. To that end, we use one of the data traces (Downlink direction of AT&T's LTE) and simulate Bernoulli packet losses with varying packet loss probabilities. Then, we normalize average throughput of each scheme to the average throughput it sees when there is no loss. This provides us good criterion to see how sensitive each scheme is to the packet losses that are not caused by congestion. Fig. 8 shows the results.

Cubic, a loss-based transport scheme, is sensitive to packet losses and considers them as congestion signals. So, when there are packet losses not due to the congestion, it suffers unwanted slowdowns. However, in parallel with normal mechanism of a loss-based scheme, C2TCP considers delay of packets as the signal of congestion too. When there are packet losses but there is no congestion (which indicates low packet delays i.e. good-condition) C2TCP can speedup the increment process of congestion window using equation 1 and rectify the unwanted slowdowns. Sprout and Verus both experience decrease in performance specially in high packet losses. However, similar to C2TCP, Vegas and BBR show very good resiliency to packet losses. This is because they both use minimum delay of packets as an extra input for calculating the sending rate, though by using different mechanisms.

E. Impact of Target and Interval

In this section, we investigate the impact of the only two parameters of C2TCP namely *target* and *Interval* on the performance of C2TCP. We use data trace of downlink direction of AT&T's LTE network for the evaluations of this section.

1) Target: Here, we set the *interval* to $100ms$ and change the *target* from $50ms$ to $100ms$. The average delay and throughput achieved for each setting has been shown in Fig. 9. As expected, by changing *target*, an application can achieve a very good balance between throughput and delay. Based on the requirements of different applications, they can change target via a socket option field. For a balanced performance, we recommend using a target value between $2\times$ to $3\times$ of RTT

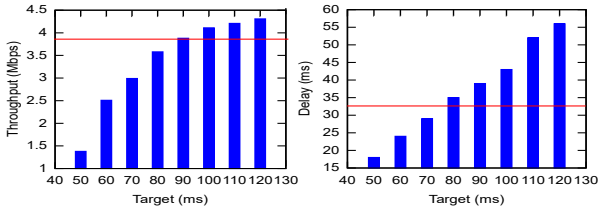


Fig. 9. Impact of Target Values on Throughput (Left) and Delay (Right). (Red lines shows performance of Codel+Cubic scheme)

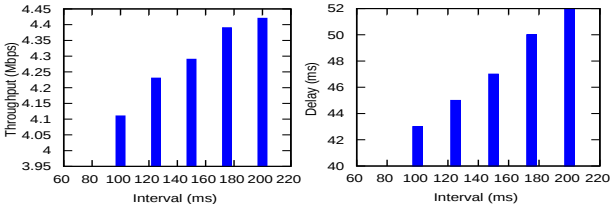


Fig. 10. Impact of Interval Values on Throughput (Left) and Delay (Right)

(here, $RTT = 40ms$). This provides enormous flexibility to applications when it is compared to in-network schemes such as RED and CoDel in which a set of queue parameters are set for all applications. Red lines in Fig. 9 show throughput and delay performance of CoDel when it is used in combination with Cubic for the same scenario. As Fig. 9 illustrates, C2TCP can be tuned to outperform the performance of CoDel, an in-network approach.

2) Interval: Now, we set the *target* to $100ms$ and change the *interval* from $75ms$ to $200ms$, and report the average delay and throughput for each setting in Fig. 10. As expected, increasing *interval* increases throughput at cost of delay and vice versa. Generally, we find out that setting *interval* to a few times of RTT is sufficient, though applications can change it using socket options, if they need.

V. Related Work

End-to-end congestion control protocols: Congestion control is always one of the hottest topics with huge studies including numeric variants of TCP. TCP Reno [24], TCP Taho [25], and TCP NewReno [11] were among early approaches using loss-based structures to control the congestion window. TCP Vegas [12] tries to do congestion control directly by using measured RTTs. TCP Cubic [8] changes incremental function of the general AIMD-based congestion window structure, and Compound TCP [29] maintains two separate congestion windows for calculating its sending rate. BBR [13] estimates both maximum bottleneck bandwidth and minimum RTT delay of the network and tries to work around this operation point, though [9] has proved that no distributed algorithm can converge to that operation point. Also, LEDBAT [27], BIC [30], and TCP Nice [31] can be mentioned among other variants. However, all these schemes are mainly designed for a wired network, i.e. fixed link capacities in the network. In that sense, they are not suitable for cellular networks where link capacity changes dynamically and stochastic packet losses exist.

Among the state-of-the-art proposals targeting cellular networks, Sprout [2] and Verus [4] are worth being mentioned. Sprout introduces a stochastic forecast framework for predicting the bandwidth of cellular link, while Verus tries to make a delay profile of the network and then use it to calculate congestion window based on the current network delay. We have compared C2TCP with most of these schemes in section IV.

AQM schemes and feedback-based algorithms: Active queue management schemes (such as RED [16], BLUE [32], and AVQ [33]) use the idea of dropping/marking packets at the bottleneck links so that end-points can react to these drops later and control their sending rates. It is already known that automatically tuning parameters of these schemes in network is very difficult [2], [14]. To solve that issue, CoDel [14] proposes using sojourn time of packets in a queue instead of queue length to drop packets and indirectly signal the end-points. However, even this improved AQM scheme still has an important issue inherited from its legacy ones: these schemes all seek a "one-setting-fits-all" solution, while different applications might have different throughput or delay constraints. Even one application can have different delay/throughput requirements during different periods of its life time.

Also, there are different schemes using feedback from network to do a better control over sending window. Among them, various schemes using ECN [34] as the main feedback. Most recent example is DCTCP [35] which changes congestion window smoothly using ECN feedback in datacenter networks. However, DCTCP similar to other TCP variants is mainly designed for stable links but not highly variable cellular links.

AQM and feedback-based schemes have a common problem: they need changes in the network which is not desirable by cellular network providers due to high CAPEX costs. Inspired by AQM designs such as CoDel and RED, C2TCP provides an end-to-end solution for this issue. Our approach doesnt require any change/modification/feedback to/from network

VI. Discussion

1) Abusing the parameters: Misusing a layer 4 solution and setting its parameters to get more share of the network by users is always a concern. For instance, a user can change the initial congestion window of loss-based schemes such as Cubic in Linux kernel. Similarly, users can abuse the Target/Interval parameters of C2TCP. Although providing mechanisms to prevent these misuses is beyond the scope of this paper, we think that setting minimum and maximum allowed values for C2TCP's parameters can alleviate the issue. In addition, in TCP, sender's congestion window will be capped to the receiver's advertised window (RcvWnd). Therefore, even by setting the Target value to a very large number in C2TCP, congestion window will be capped to RcvWnd at the end.

2) C2TCP flows with different requirements on one user: When a cellular phone user runs a delay sensitive application (such as real-time gaming, video conferencing, virtual reality

content streaming, etc.), flow of that application is the main interested flow (highest priority one) for the user. Therefore, through the paper, we have assumed that it's rare to have more flows competing with that highest priority flow for the same user. However, in case of having multiple flows with different requirements for the same user, we think that any transport control solution (such as Cubic, Vegas, Sprout, C2TCP, etc.) should be accompanied with prioritization techniques at lower layers to get good results in practice (e.g. [36], [37]). For instance, one simple existing solution is using the strict priority tagging for packets of different flows (by setting differentiated services field in the IP header) and later serve flows based on these strict priorities in the network.

3) Setting Target in practice: In practical scenarios, instead of setting Target value per application, we could set it per class of applications. In other words, we could let applications choose their application types. Then, C2TCP would set the Target using a table including application types and their corresponding Target values made in an offline manner.

VII. Conclusion

We have presented C2TCP, a congestion control protocol designed for cellular networks to achieve low delay and high throughput. C2TCP's main design philosophy is that achieving good performance does not necessarily comes from complex rate calculation algorithms or complicated channel modelings. C2TCP attempts to absorb dynamics of unpredictable cellular channels by simply investigating local minimum delay of packets in a moving time window. Doing that, C2TCP stands on top of an existing loss-based TCP and provides it with a sense of delay without using any network state profiling, channel prediction, or complicated rate adjustments mechanisms. We show that C2TCP outperforms well-known TCP variants and existing state-of-the-art schemes which use channel prediction or delay profiling of network.

References

[1] (2017) State of the internet. [Online]. Available: https://www.akamai.com/fr/fr/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf

[2] K. Winstein *et al.*, "Stochastic forecasts achieve high throughput and low delay over cellular networks." in *NSDI*, 2013, pp. 459–471.

[3] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *ACM SIGCOMM CCR*, vol. 43, no. 4. ACM, 2013, pp. 123–134.

[4] Y. Zaki *et al.*, "Adaptive congestion control for unpredictable cellular networks," in *ACM SIGCOMM CCR*, vol. 45, no. 4. ACM, 2015, pp. 509–522.

[5] J. Huang *et al.*, "An in-depth study of lte: effect of network protocol and application behavior on performance," in *ACM SIGCOMM CCR*, vol. 43, no. 4. ACM, 2013.

[6] H. Jiang *et al.*, "Tackling bufferbloat in 3g/4g networks," in *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012, pp. 329–342.

[7] W. L. Tan *et al.*, "An empirical study on 3g network capacity and performance," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 1514–1522.

[8] S. Ha *et al.*, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[9] J. Jaffe, "Flow control power is nondecentralizable," *IEEE Transactions on Communications*, vol. 29, no. 9, pp. 1301–1306, 1981.

[10] R. Netravali *et al.*, "Mahimahi: A lightweight toolkit for reproducible web measurement," 2014.

[11] T. Henderson *et al.*, "The newreno modification to tcp's fast recovery algorithm," Tech. Rep., 2012.

[12] L. S. Brakmo *et al.*, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, vol. 24, no. 4.

[13] N. Cardwell *et al.*, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[14] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[15] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[16] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 4, pp. 397–413, 1993.

[17] "Mobile edge computing introductory technical white paper," etsi.org, Tech. Rep., 2014. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1\%2018-09-14.pdf

[18] S. Sesia *et al.*, *LTE-the UMTS long term evolution: from theory to practice*. John Wiley & Sons, 2011.

[19] M. Mathis *et al.*, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM CCR*, vol. 27, no. 3, pp. 67–82, 1997.

[20] V. Jacobson *et al.*, "Red in a different light."

[21] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[22] (2017) Maxcdn a mobile content delivery network solution. [Online]. Available: https://www.maxcdn.com/solutions/mobile/

[23] G. Appenzeller *et al.*, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, 2004.

[24] D. Cox and L.-R. Dependence, "a review," *Statistics: An Appraisal, HA David and HT David (Eds.)*, pp. 55–74, 1984.

[25] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM CCR*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[26] (2017) M-cord: Mobile cord. [Online]. Available: opencord.org/wp-content/uploads/2016/03/M-CORD-March-2016.pdf

[27] D. Rossi *et al.*, "Ledbat: The new bittorrent congestion control protocol." in *ICCCN*, 2010, pp. 1–6.

[28] (2017) Mobile/tablet operating system market share. [Online]. Available: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

[29] K. Tan *et al.*, "A compound tcp approach for high-speed and long distance networks," in *Proceedings-IEEE INFOCOM*, 2006.

[30] L. Xu *et al.*, "Binary increase congestion control (bic) for fast long-distance networks," in *Proceedings-IEEE INFOCOM*, vol. 4. IEEE, 2004, pp. 2514–2524.

[31] A. Venkataramani *et al.*, "Tcp nice: A mechanism for background transfers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 329–343, 2002.

[32] W.-c. Feng *et al.*, "The blue active queue management algorithms," *IEEE/ACM transactions on networking*, vol. 10, no. 4, 2002.

[33] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management," in *ACM SIGCOMM CCR*, vol. 31, no. 4. ACM, 2001, pp. 123–134.

[34] S. Floyd, "Tcp and explicit congestion notification," *ACM SIGCOMM CCR*, vol. 24, no. 5, pp. 8–23, 1994.

[35] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *ACM SIGCOMM CCR*, vol. 40, no. 4. ACM, 2010, pp. 63–74.

[36] S. Abbasloo *et al.*, "Hyline: a simple and practical flow scheduling for commodity datacenters," in *IFIP Networking Conference (IFIP Networking) and Workshops, 2018*. IEEE, 2018.

[37] C.-Y. Hong *et al.*, "Finishing flows quickly with preemptive scheduling," in *ACM SIGCOMM CCR*. ACM, 2012, pp. 127–138.

The Virtue of Gentleness: Improving Connection Response Times with SYN Priority Active Queue Management

Tristan Braud*, Martin Heusse¶, and Andrzej Duda¶

{braudt@ust.hk, martin.heusse@imag.fr, andrzej.duda@imag.fr}

*Department of Computer Science and Engineering, Hong Kong University of Science and Technology.
¶Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France.

***Abstract*—We have analyzed network traces of TCP connections and observed that there are many more losses during the handshake than for the remainder of the data exchange. Although recently developed AQM schemes can efficiently reduce latency related to *bufferbloat*, only more complex solutions relying on Fair Queueing (FQ) can improve the long delays resulting from the loss of a packet during the establishment of a TCP connection. In this paper, we propose SPA (SYN Priority Active queue management), a new low-complexity queue management scheme that combines the benefits and simplicity of the most recent AQM schemes while achieving performance comparable to more complex combinations of Fair Queueing and AQM. Our evaluation shows that the SPA performance is close to FQ CoDel for only a fraction of the complexity and resource usage.**

I. Introduction

Shortening the response time and reducing overall latency of TCP transfers is paramount to improve the responsiveness of information access over the network. In this paper, we focus on the connection establishment phase, which is often the limiting factor of the transfer response time, because at the beginning, connections do not have an estimate of the round trip time (RTT) between the client and the server, so they use a long default retransmission timeout (RTO). The reception of SYN/ACK immediately updates RTO so that all other segments can be quickly resent, even without Fast Retransmit. Thus, the client can only recover from the loss of the initial SYN segment or the corresponding SYN/ACK after a long period several orders of magnitude larger than the recovery time of subsequent data segments. This effect is particularly detrimental to short connections. For larger ones, the impact of a SYN loss becomes less significant, especially in the case of non-interactive traffic. Despite the importance of the connection establishment phase, speeding it up has received less attention than ensuring the good performance of the bulk data exchange [1]. Over the past years, many proposals aimed at maximizing link utilization and achieving low delays, the most well-known being *CoDel* [2] and *FQ CoDel*, its Fair Queueing counterpart [3]. However, the most recent packet scheduling schemes [2], [4] present the drawback of either increasing SYN drops, or relying on Fair Queueing, which results in higher CPU and memory usage.

To evaluate the impact of SYN or SYN/ACK losses on the transfer response time, we have analyzed a set of publicly available real world traffic traces. The analysis shows that SYN and SYN/ACKs are generally lost much more often than other TCP segments, which is another reason to consider them in a particular way. We would expect that the SYN retransmission rate is twice that of regular data segments, because it happens after a SYN loss and also after a SYN/ACK loss. In fact, we observe an even much higher SYN retransmission rate: several times the retransmission rate of all segments, which exacerbates the performance problem for short connections. SYN/ACK losses has a detrimental impact on performance: their retransmissions cause a two-fold increase of the retransmission delays experienced by connections. Based on these observations, we propose SPA (SYN Priority Active queue management), a queueing scheme with two packet queues both managed by *CoDel* (Controlled Delay Management) [2]: a higher priority queue handles SYN and FIN packets, and a lower priority one manages data segments. SPA is thus a pair of queues with independent AQM mechanisms. It is much less complex than any fair queueing scheme and results in short connection setup times and smooth low latency data transmission. Similarly to CoDel and FQ CoDel, this scheme is designed to be deployed at the "last mile" of the network, typically in home routers, where resources (CPU and bandwidth) tend to be scarce, and reactivity is key.

The contribution of this work is threefold: first, we analyze network traces to get insight into the behavior of the TCP establishment phase and observe a significantly higher SYN retransmission rate compared to other types of segments. Second, we model the effect of SYN retransmissions on transfers of a limited size to show that SYN losses account for a large part of the response time. Finally, we evaluate SPA through measurements on a testbed and compare its performance with recent and well-established scheduling mechanisms. We show that under normal traffic conditions, SPA performs similarly to FQ CoDel, for a fraction of complexity. We also shed light on a case for which FQ Codel collapses due to its own design.

II. Related Work

The delay introduced by the connection establishment phase is often overlooked in the literature even in recent papers [5], [6]. Some authors observed that the loss probability of SYN segments and regular ones may differ [7], although without exploring the impact of this difference. Ciullo et al. [8] observed the distribution of the connection completion time and realized that more than 70% of dropped packets

ISBN 978-3-903176-08-9

are recovered after RTO. They also proposed two schemes to overcome long recovery delays due to the loss of the last data segment in a flow. Anelli et al. [1] made a case for protecting SYN segments to improve connection response times. They introduced *REDFavor*, a RED mechanism with a specific processing of SYN segments. Another idea proposed in the literature is to resend aggressively SYNs to avoid the impact of the long timeout on lost SYN segments [9]–[11]. Although functional, this last solution requires the user to modify the operating system or use additional software, while adding some traffic in an already congested link. Similarly, solutions such as WonderShaper [12] or DD-WRT [13] also take the approach of giving a higher priority to all control messages. However, they rely on FIFO queues that requires prior configuration to achieve low latency. This approach cannot be efficiently used in delay or bandwidth-varying environments.

Recently, researchers have started to discuss the *bufferbloat* effect, the problem of long delays due to excessive buffer sizes in access networks [14]. A workgroup on bufferbloat began in 2011 [15] and some analyses of the problem started to appear [16]–[19]. Two main proposals for solving the problem include *CoDel* [20] and *PIE* (Proportional Integral controller Enhanced) [4], [21]. CoDel measures the packet sojourn time in the queue and drop packets at the queue head to keep the delay from exceeding a reasonable value for any significant period of time. It requires per packet timestamps, but it does not need any configuration parameter except for the default *threshold delay*. *FQ (Fair Queueing) CoDel* [3] extends CoDel with the principles of *Stochastic Fair Queueing* [22] to manage per flow queues and mitigate the adverse effects of purely random packet drop. PIE [21] controls the average queueing latency so it stays at a reference value. It combines the benefits of both RED and CoDel: PIE randomly drops a packet at the beginning of the congestion detected based on the queueing latency like CoDel. PIE can ensure low latency and achieve high link utilization under various congestion conditions. Schwardmann et al. evaluated by simulation the robustness of CoDel, PIE, and ARED for various static and dynamic scenarios [23] in a simple set-up with one link and a variable number of bulk TCP flows. *ARED* (Adaptive RED) is an active queue management scheme that attempts to stabilize the average queue size around some preset target queue size [24]. PIE achieved lower delays than CoDel and ARED for low capacity links, but for higher capacity links, CoDel and ARED resulted in lower delays. In dynamic scenarios, CoDel results in a lower maximum delay than the other schemes. Khademi et al. evaluated CoDel, PIE, and ARED [25], but in an experimental setup using Linux implementations in a wired testbed for bulk TCP transfers. The authors also observed that the CoDel "dropping-mode" interval needs to be set lower than the default value, while we note in this paper (see Section VI-B), the noticeable influence of the target delay that it uses in the case of the FQ CoDel variant. These results contradict the claim that CoDel is a parameterless AQM. Otherwise, ARED performed the best in the Khademi et al. study except when the number of flows on the bottleneck link was very small.

III. Analysis of Real World Traces

To evaluate the impact of SYN or SYN/ACK losses, we have analyzed a set of publicly available traffic traces: five

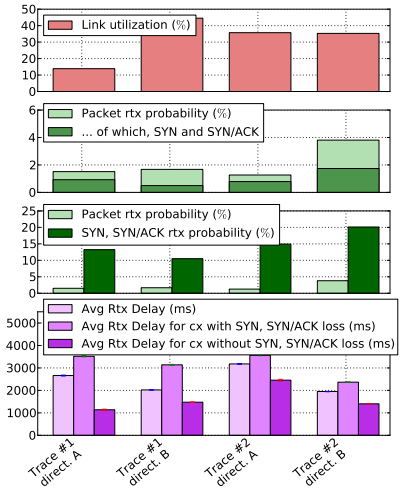


Fig. 1. Analysis of the CAIDA traces in both directions ordered by ascending order of link utilization in direction A. SYN losses represent a large part of all losses and SYN segments have a significantly higher loss probability than data segments leading to a significant impact on retransmission delays.

sets of traces from the CAIDA data set[1] recorded on two different days at an Equinix datacenter in Chicago connected to a 10 Gigabit Ethernet backbone link of a Tier 1 ISP. As the traces from CAIDA are split according to the direction, we analyze them in one direction at a time. We consider two days with differing traffic load patterns: for the trace taken on March 20th, 2014 the link utilization in direction A is markedly lower than on September 18th, 2014. For readability purposes, we will further refer to the traces as Trace "Real" and Trace "Short" respectively. All measurements except the link utilization concern connections with at least one SYN, which represents from 180,000 to 650,000 connections per trace. We have filtered out the connections with retransmission delays greater than 45s to remove inconsistent data from the analysis (the value of 45s corresponds to the total delay after loosing 3 SYNs at the start of a connection due to the exponential backoff[2]). Due to the number of samples, the 95% confidence interval could not be represented for loss probabilities and is barely visible for retransmission times.

Figure 1 shows two striking phenomena:

1) SYN and SYN/ACK retransmissions account for a large part of the retransmissions (2^{nd} plot)
2) the probability of SYN or SYN/ACK retransmissions is exceptionally high—between 10% and 20% (3rd plot). The connections with SYN or SYN/ACK retransmission suffer from an average retransmission delay almost twice that of other connections (4th plot).

Note that the initial RTO is set to 3 seconds by default on Windows, FreeBSD, and Linux (prior to 2011; it is now 1 s). 10% to 20% of all connections are affected whereas the packet loss rate remains limited at 1% to 3.5%.

[1]The CAIDA UCSD Anonymized Internet Traces 2014 *20/03/2014 12:59:11, 13:03:00, 13:06:00 and 18/09/2014 13:07:00, 13:19:00* http://www.caida.org/data/passive/passive_2014_dataset.xml

[2]as stated in the FreeBSD source code *"the odds are that the user has given up attempting to connect by then."* [26].

TABLE I. ANALYSIS OF MAWI TRACES

	Dir A	Dir B
Packet rtx prob.	1.65%	2.73%
...of which, SYN, SYN/ACK represent	29.8%	79.5%
SYN and SYN/ACK rtx prob.	13.1%	29.9%
Average rtx delay (ms)	5276.96	2114.93
Average rtx delay for connections with SYN or SYN/ACK loss (ms)	6324.93	2390.63
Average rtx delay for connections without SYN or SYN/ACK loss (ms)	4707.11	1011.93

TABLE II. MAWI TRACES, SYN FLOOD FILTERED OUT

	Dir A	Dir B
Packet rtx prob	1.63%	2.72%
...of which, SYN, SYN/ACK represent	28.8%	78.9%
SYN and SYN/ACK rtx prob	12.8%	40.6%
Average rtx delay (ms)	5312.9	2114.93
Average rtx delay for cx with SYN or SYN/ACK loss (ms)	6469.4	2390.63
Average rtx delay for cx without SYN or SYN/ACK loss (ms)	4710.5	1011.93

The higher retransmission rate of SYNs compared to other segments is expected as a SYN/ACK drop in the reverse direction always causes a timeout, whereas cumulative ACKs make the established connections relatively immune to losses on the return path [27]. So, as long as SYN and SYN/ACK segments are as likely as other packets to be dropped, the retransmission rate doubles. Since the queues in many routers are managed as *packet FIFOs*, small SYN packets are just as likely to be dropped as the larger ones. On the contrary, a *byte-based FIFO* would favor small packets that can generally still fit in the queue when larger packets are dropped. In our trace analyses, we do not see any situation in which small packets would be retransmitted less than larger ones, which means that most routers use packet FIFOs. (If some routers use byte FIFO, it is beneficial to small flows as shown below.) However, the fact that retransmissions occur for losses in both directions only explains a small fraction of the observed difference. Another explanation of the discrepancy between SYN and regular segment losses could be TCP congestion control algorithms. They are designed so that connections loose a limited number of segments per congestion episode (1 for the congestion avoidance phase). After a loss, TCP divides the congestion window by 2 to reduce the transmission rate and delay the next occurrence of congestion. Subsequent packets are thus less likely to experience another loss. In contrast, SYN packets arrive at random and their potential retransmission is so distant in time that the conditions are uncorrelated, which could lead to more losses than regular packets. Again, more investigations are required to explain this difference.

To confirm the findings, we have also analyzed traces collected by the MAWI Working Group of the Wide project on the sample point F (1Gb/s transit link between WIDE and an upstream ISP). Again, we filter out connections with retransmission delays greater than 45s (see Table I). We can first observe that one direction presents less losses than the other one although, according to the retransmission delays, connections experience many timeouts. We have observed the same phenomenon in the CAIDA traces taken on September 18, 2014 in direction A: although losses are less frequent than in direction B, most of them end up as a timeout. The reason may be bufferbloat in an upstream buffer that causes higher delays, thus setting off the RTO timer. The observed delays corroborate the results from the CAIDA traces: SYN and SYN/ACK losses account for almost half of the losses. SYN and SYN/ACK retransmissions cause a two-fold increase of the retransmission delays experienced by connections.

As we analyze connection establishment, one concern is that the presence of SYN flood attacks may bias the results. We have filtered out potential SYN floods by removing

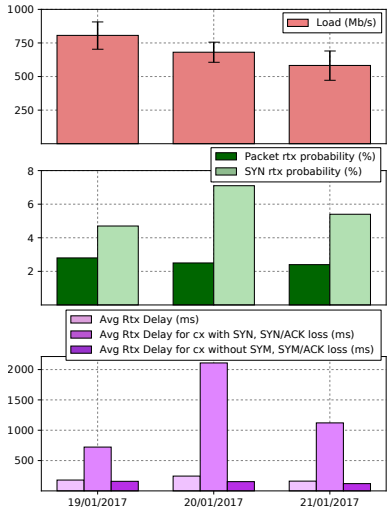


Fig. 2. Analysis of the bidirectional MAWI traces ordered by date for connections with a full three way handshake and a data packet. Although SYN losses represent a lower part of total losses, they still have a higher loss probability than data segments leading to a significant impact on retransmission delays.

connections involving IP addresses that show the signs of a SYN flood, which is possible for bidirectional MAWI traces. To identify them, we have first counted the connections that effectively carry data segments for each IP address present in the trace. We have filtered hosts that showed strong signs of being under a SYN flood attack (or could have been dysfunctional), when they experienced 1000 times more connections without data than with data (with a minimum of 1000 connections without data). We have removed 69 out of 587,262 IP addresses present in the trace, which results in removing over 30% of connections mainly in direction B. Table II presents the corrected data. In direction A, we have removed only a few connections and the results are mostly unchanged. Removing a large part of connections in the reverse direction dramatically increases the SYN retransmission probability up to 40%, while the delays and the retransmission probabilities are not significantly affected. Indeed, in the case of a SYN flood, we expect to see a large amount of unique SYNs, without any further segment or retransmission, as the attacker does not have any interest in establishing a real connection. As the MAWI traces are bidirectional, we can now focus on the analysis of connections that manage to pass the three-way handshake and transmitted at least one data segment. Figure 2 presents the results. Fully established connections present more contrasted results. Indeed, the SYN loss probability is now only twice as big as the packet loss probability, which

corroborates the hypothesis we have made earlier, stating that we should observe a SYN loss probability twice the packet loss probability as the loss of a SYN/ACK packet will trigger a SYN retransmission from the other side. Although we observe less SYN losses, their impact on the average retransmission time is much higher: between 10% and 40% of the average retransmission time is due to connections with at least a SYN loss. Such connections also present average retransmission times 4 to 14 times higher than connections without a SYN loss. Our findings from both data sets are in line with the results by Damjanovic et al. in the LBNL/ICSI Enterprise Tracing Project [9] that showed an overall 10% of SYN retransmissions for all connections in a LAN and 2% of SYN retransmissions for successful connections. Still, we observe much higher loss rates (10–20% overall SYN loss and 5–8% for successful connections). This difference could be explained by the fact that the CAIDA and MAWI traces come from a transit link where it is more likely to include 3/4G, WLANs, or even satellite traffic with variable delays and random losses, or simply more congestion along the way. Such conditions are more prone to generate losses, create timeouts, and in general, create more fluctuations in the analyzed values.

IV. Model of the TCP Response Time under SYN Losses

To evaluate how SYN losses impact short TCP transfers, we propose to estimate the transfer duration based on the well-known model proposed by Mathis et al. [28] and Padhye et al. [29], extended to take into account SYN retransmissions.

We first consider the congestion avoidance phase of a TCP connection with constant *RTT* and subject to packet loss probability p_l. The congestion window oscillations are equivalent to cyclic oscillations between $\frac{\bar{W}_{\max}}{2}$ and $\bar{W}_{\max}$ so the average expected window is:

$$\bar{W}_{\text{avg}} = 3/4 \times \bar{W}_{\max}$$

and the expected throughput in pkt/s is the following:

$$\bar{X} = \frac{3}{4} \times \frac{\bar{W}_{\max}}{RTT} \approx \frac{1}{\text{RTT}} \sqrt{\frac{3}{2\,p_l}},$$

using Mathis' estimate for $\bar{W}_{\max}$. We model the slow start phase by considering an initial congestion window of one segment going up to $\bar{W}_{\max}$. We approximate the number n_{ss} of RTT taken by the slow start phase as $2^{n_{\text{ss}}-1} = \bar{W}_{\max}$, so that

$$n_{\text{ss}} = \frac{log(\bar{W}_{\max})}{log(2)} + 1.$$

Now, we need to estimate the time spent in recovery of losses that may happen during the connection. For one loss per congestion event that happens with probability p_l and recovery time R_t, a connection of size S MSS spends time $t_{\text{recovery}} = S \times R_t \times p_l$ in recovery, knowing that optimistically, $R_t \approx RTT$. If the probability of a SYN loss is p_{Sl}, the time spent in the connection establishment phase is:

$$t_{\text{syn}} = RTT + RTO_0 \sum_{k=1}^{\infty} (p_{\text{Sl}})^k = RTT + \left(\frac{1}{1-p_{\text{Sl}}} - 1 \right) RTO_0, \quad (1)$$

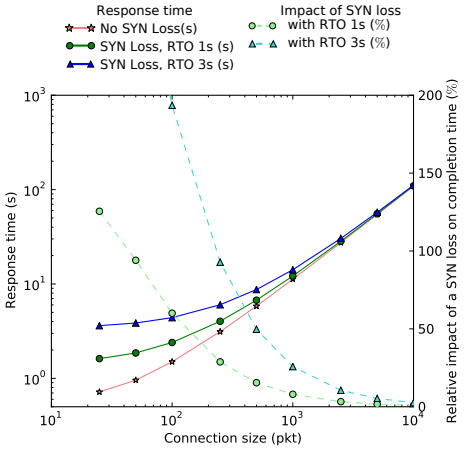


Fig. 3. Response time of connections with and without SYN loss

where RTO_0 is initial retransmission timeout. We can approximate p_{Sl} with $\approx 2p_l$ or obtain it from measurements.

For a connection of size S, the total response time is:

$$t_{tot} = t_{\text{syn}} + t_{\text{slow\_start}} + t_{\text{congestion avoidance}} + t_{\text{recovery}} + t_{\text{fin}}, \quad (2)$$

where $t_{\text{slow\_start}} = n_{\text{ss}} \times RTT$, $t_{\text{congestion\_avoidance}} = \frac{S-(2^{n_{\text{ss}}}-1)}{\bar{X}}$ and $t_{\text{fin}} = RTT$.

Figure 3 represents the estimated response time of connections with and without SYN loss with RTO of 1 s or 3 s, for the connection size between 25 and 10000 segments in the conditions of Trace "Real" (Figure 1). A SYN loss is a major problem for short connections as it increases the response time by up to 200%. The impact becomes negligible (i.e., $< 10\%$ of the response time) for connections larger than 1000 segments for RTO of 1 s and 2000 segments for RTO of 3 s. Short connections, which are generally already penalized compared to long connections, dramatically suffer from this phenomenon.

V. SPA – SYN Priority AQM Scheme

CoDel [2], an almost parameterless, delay-based queueing management scheme, is a recent solution to bufferbloat. However, as other AQMs designed to tackle excessive queueing delays, CoDel assumes that all losses are good to the network, as long as they contribute to keeping the delay low. Yet, many losses will always result in timeouts, especially head drops—when the initial TCP RTO is still up to make matters worse—and tail drops can also result in long timeouts if the connection presents some delay fluctuations. FQ CoDel [3] solves this problem by giving a higher priority to the first packets, but at the cost of a higher complexity and memory footprint: for instance, the default Linux implementation uses 1024 separate queues. Our approach is to strike a compromise: we manage only two packet queues by CoDel—a higher priority queue for SYN and FIN packets and a lower priority one for regular packets, and do not keep track of connection states. The pseudocode below defines the scheme more formally:

SPA (SYN Priority AQM) Scheme

```
synfinqueue = CoDel()
packetqueue = CoDel()
def enqueue(p):
```

```
    if p.flag.SYN==set or p.flag.FIN==set:
      synfinqueue.enqueue(p)
    else:
      packetqueue.enqueue(p)
  def dequeue(p):
    if synfinqueue.is_empty():
      packetqueue.dequeue(p)
    else:
      synfinqueue.dequeue(p)
```

Three types of losses may lead to a timeout: SYN, FIN, and tail losses. While it is difficult to detect tail losses at the router level, it is much easier to isolate SYN and FIN in a separate queue and avoid timeouts. The queue for SYN and FIN has priority over the second one for data packets and both queues are managed by CoDel. In this way, we significantly increase the overall performance by preventing most of timeouts, while keeping complexity much lower than FQ CoDel. We can easily implement SPA based on an existing CoDel implementation in coordination with regular system tools. For our experiments, we have set up SPA with a priority queue serving two CoDel subqueues using `tc` [30] and `iptables` for packet filtering. As mentioned before, another possibility to avoid SYN and SYN/ACK losses is to dimension the buffer in bytes rather than in packets. Although effective and relatively straightforward to implement, this solution still presents one major drawback: it requires to know in advance the link capacity, whereas in many cases, it is unknown and fast varying, as for WiFi or cellular networks. Excessive queueing space—bufferbloat—results in high latency in the network or conversely, an insufficient buffer may result in low link utilization. By design, the proposed SPA scheme does not suffer from such limitations.

As the SPA scheme favors SYN packets over data packets, it may raise some security concerns. We do not want an attacker exploit our solution to favor her/his connections or to accelerate SYN DoS attacks. Regarding the first issue, a basic attack would be to set the SYN or FIN flag on every data packet. Admitting that the receiver does not discard such packets, checking the packet length is enough to prevent abuse. SYN DoS attacks raise a more complex issue. Indeed, as SPA favors SYN packets, a router could be used as a relay to accelerate the transit of SYN packets at the bottleneck and amplify a SYN flood attack. However, similarly to CoDel and FQ CoDel, SPA is intended to be deployed at the bottleneck, usually at the last mile of the network, where an attacker may already have full control of the link. For instance, deploying SPA in home routers is harmless. Most Fair Queueing solutions are also vulnerable to this type of exploit, as they prioritize packets from new connections, since they constantly try to equalize the service received by all connections, and the new one initially shows a deficit. A batch of SYN packets with various source addresses and ports would typically be favored over data packets from existing connections. We show this phenomenon in Section VI-B, where FQ CoDel gets swarmed by a large number of new connections, resulting in abnormal behavior, whereas SPA proves to be more resilient.

VI. Testbed Experiments and Performance Comparisons

We have run a series of experiments on a testbed network composed of three computers (cf. Figure 4). One of them acts as a router interconnecting two others, while the two

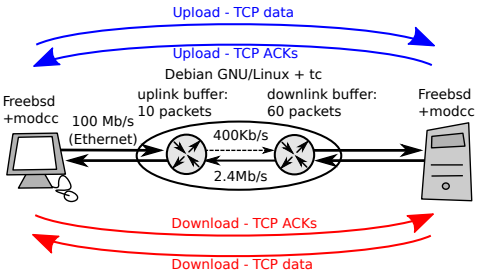


Fig. 4. Testbed network for experiments. The central computer emulates the bottleneck link in both directions and implements the considered policies. The traffic is in the downlink direction with or without a long lasting upload TCP data transfer.

ends are used as a client and a server. This simple topology allows observing the behavior of various queueing policies at the bottleneck. The hosts run FreeBSD 9.3 that allows us to test the latest variants of TCP without switching operating systems. The router runs Debian GNU/Linux (wheezy) with a slightly modified 3.18 kernel as described below. On top of this operating system, we run the `tc` tool to emulate links and various queueing management policies. We have set the kernel context switching frequency to 1kHz to emulate the bottleneck link smoothly for the bit rates up to 10Mb/s. We use the `ipmt` suite [31] to generate connections as follows: connection arrival times follow a Poisson process with parameter λ and connection sizes follow a Zipf distribution with parameter μ. We have run experiments with and without reverse traffic.

TABLE III. Experiment parameters

Uplink		
Capacity C_u	0.4Mb/s	2Mb/s
Delay D_u	52ms	
Scheduling Q_u	Packet FIFO	
Downlink		
Capacity C_d	2.4Mb/s	10Mb/s
Delay D_d	42ms	
Scheduling Q_d	Packet FIFO - Byte FIFO RED - ARED - REDFavor CoDel - FQ CoDel – PIE SFQ SYN Prio - SYN/FIN Prio - SPA	
Connection Generation Parameters		
Traffic Type	**"Real"**	**"Short"**	**"10M"**
$1/\lambda$ (s)	0.12	0.017	0.032
μ	1.7	2	1.7

We run experiments under three different traffic conditions shown in Table III. As SPA targets the last mile section of a network, we consider the case of typical ADSL link as it is still far more deployed than fibers [32]. We use a 42 ms round trip delay, which corresponds to the ADSL interleaving delay (typically 24 ms) plus an intra-continental propagation time [33]. The slightly higher uplink delay helps to avoid perfect synchronization between the upload and download feedback loops. The traffic conditions correspond to a high link utilization of 90%. By varying the λ and μ parameters as well as the connection size, we obtain various traffic conditions:

Traffic Type "Real" corresponds to realistic assumptions with connection sizes distributed according to a heavy tailed Zipf distribution: $1/\lambda = 0.12s$, $\mu = 1.7$, which leads to an average

size of 21 segments per connection, a median of 2 packets and 80% of connections spanning 6 segments or less. This load is applied with and without reverse traffic.

Traffic Type "Short" represents extreme conditions to push the tested schemes to the limits. We generate a connection every $1/\lambda = 0.017s$, connections have the average size of 3 segments ($\mu = 2$), a median size of 1 segment, and the maximum size of 100 segments (80% are 4 segments or less)

Traffic Type "10M". With this type, we analyze the behavior of the tested schemes for a link with the increased capacity to 10Mb/s with 95% load composed of connections arriving on the average every 0.032 second, with an average size of 26 segments and a median of 2 segments (80% under 7 segments).

We test the following scheduling schemes at downlink queue Q_d: regular FIFO (limited either in packets – Packet FIFO or in bytes – Byte FIFO), AQM (RED, ARED [24]), delay controlled mechanisms (CoDel and PIE [4]), Fair Queueing (FQ CoDel and SFQ – Stochastic Fair Queueing [22]). We compare these queueing schemes to priority queues with a higher priority given to SYN or SYN/FIN (SYN Prio, SYN/FIN Prio, RedFavor [1], and SPA). The uplink queue remains unchanged for all experiments (Packet FIFO). We use the default threshold delay of 5 ms for CoDel, FQ CoDel, and SPA. As we have discovered that in some cases, this parameter needs to be tweaked for CoDel and FQ CoDel, we also try the value of 20 ms. We did not include solutions at the edge like setting the SYN retransmission timeout to a more aggressive value [9]–[11], because they may only improve the retransmission time, but they cannot lower the SYN loss rate.

A. *Results for Traffic Type "Real"*

Figure 5 presents the results for Traffic Type "Real". Similarly to Figure 1, due to the number of connections (8326), we are only able to display 95% confidence intervals for the retransmission and response times. The two most prominent outcomes are: (1) all schemes provide better response times than Packet FIFO and (2) only a few schemes clearly stand out: FQ CoDel, SFQ, RedFavor, and SPA. **For short connections, FQ CoDel 20ms and SPA are the best ones**. SFQ attains the best result for longer connections with REDFavor and SPA also having short response times. We observe remarkable differences of RTT between delay-based solutions and regular queueing schemes: CoDel and PIE halve the RTT compared to FIFO or RED, while FQ CoDel and SPA achieve the lowest RTTs. We also see that about one fourth of the average retransmission delay is contributed by connection SYN losses: looking at the traces, a small fraction of connections experience a 3 second delay before even being able to send a single packet. Byte FIFO gets shorter retransmission delays (20%) than Packet FIFO, reaching the same performance as RED with almost no SYN loss and a lower RTT than Packet FIFO. **Byte FIFO even outperforms RED, ARED, CoDel, and PIE in terms of response times!** SFQ shows unusually high retransmission delays. Indeed, even if there are almost no SYN losses, most losses result in one and often multiple timeouts. In essence, the **Fair Queueing algorithm leads to the starvation of long flows**: as the queue experiences high load with many short connections, the longer ones are not serviced and eventually timeout. Yet, Fair Queueing policies

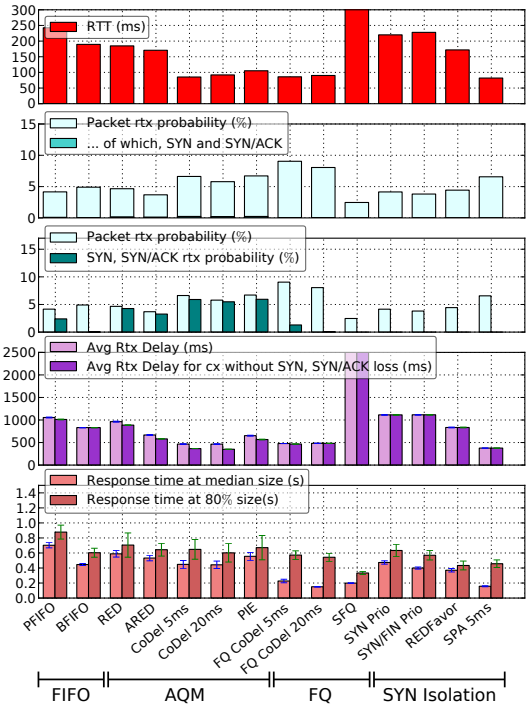


Fig. 5. Average RTT, loss percentage, retransmission delays and response times for various queueing schemes under Traffic Type "Real". FQ CoDel, SFQ, RedFavor, and SPA achieve the shortest response time.

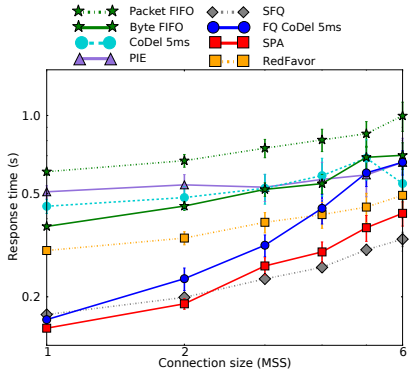


Fig. 6. Response times of connections for various queueing schemes under Traffic Type "Real". SPA and Fair Queueing schemes result in significantly lower response times compared to other AQM and FIFO.

display some of the best response times for short connections. Considering the isolation of SYN and/or FIN in a separate FIFO (SYN and SYN/FIN Prio, REDFavor), we do not observe a real improvement in RTT and retransmission times compared to basic FIFO. Nevertheless, there are fewer SYN retransmission, which improves the response times especially for short connections. Finally, **SPA results in the shortest retransmission delay**, while keeping one of the lowest average RTT and response times. FQ CoDel achieves a similar though slightly worse performance if left with default parameters.

Figure 6 presents the connection response times for a selection of the considered policies. There is a manifest separation between two groups: in the first one, PIE and CoDel exhibit similar performance, while **Byte FIFO shows a noticeable improvement compared to Packet FIFO** (connections complete

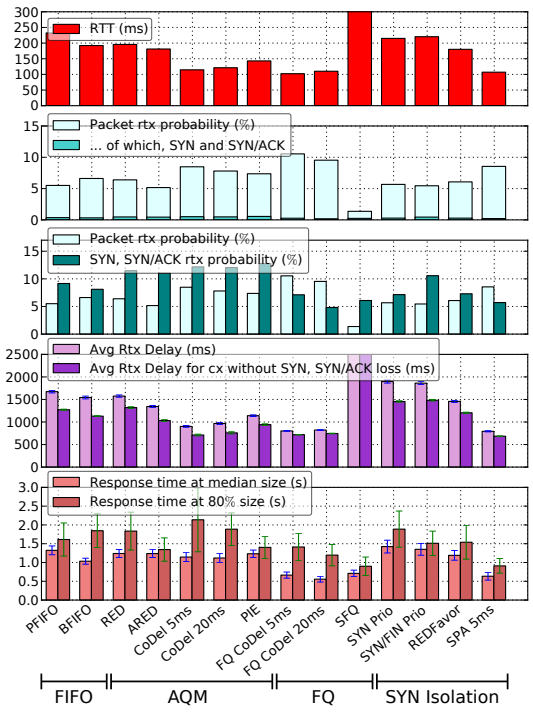


Fig. 7. Average RTT, loss percentage, retransmission delays, and response times under Traffic Type "Real" with one long reverse connection. SYN retransmissions rate, retransmission delays, and response times raise drastically. FQ CoDel and SPA achieve the shortest response times for short connections, while SFQ and SPA are the best for longer connections.

40% faster). In the second group, **SPA, SFQ, and FQ CoDel present lower response times** with an improvement between 100% and 300% for the shortest connections. In the case of delay based solutions like CoDel or PIE, this plot confirms that the philosophy of "*loss is good*" is detrimental when control segments such as SYN are not considered in a specific way: the schemes lead to high data segment retransmission rates, which is actually good for reducing congestion and delays, but also to high levels of SYN retransmissions. Consequently, the schemes experience longer retransmission delays and longer delays, even if they still manage to improve the performance over Packet FIFO. REDFavor performs better than other AQM schemes: with this simple modification, response times are halved compared to Packet FIFO, which is a quite noticeable enhancement compared to other AQM mechanisms. Finally, our solution performs better than FQ CoDel and SFQ for short connections. For connections with more than 3 segments, the difference between the regular and modified FQ CoDel becomes less significant and connections under SPA complete in 30% less time than with FQ CoDel. SFQ continues to obtain good response times similar to those obtained with SPA, but with inacceptable retransmission times for larger connections, as we can see in Figure 5.

We complement the first experiment that dealt with unidirectional traffic by adding one long reverse connection. The idea is to approach "near real" traffic conditions as encountered for instance behind an ADSL link with a user uploading files to a Cloud service provider. The results appear in Figures 7 and 8. The first expected result is an increase of SYN losses in the presence of reverse traffic. Indeed, SYN/ACKs can be

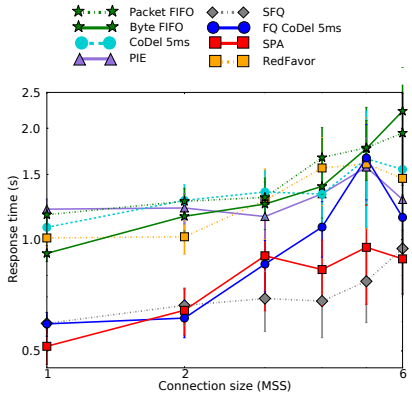


Fig. 8. Response times of connections for various queueing schemes under Traffic Type "Real" with one long reverse connection. Important gap between AQM and Fair Queueing schemes, but more differences inside each group.

lost on the return path, while the presence of ACKs in the download queue virtually reduces the queue size and leads to an overall increase of losses [34]. We observe 25% more losses on average and twice as many SYN losses. Consequently, SYN retransmissions account for a larger part of the retransmission delay: connections without a SYN loss experience retransmission delays 25% lower than average. Similarly, response times are much longer. We see that for short connections, FQ CoDel and SPA achieve the shortest response times, while SFQ and SPA are the best for the longer connections. SPA has once again similar performance to FQ CoDel, both in terms of RTT and retransmissions times, with response times halved compared to most of the AQM mechanisms.

Concerning response times (cf. Figure 8), we observe the same phenomenon: even if connections take much more time to complete compared to the situation without reverse traffic, we can still differentiate between two groups: FQ CoDel, SFQ, and SPA obtain much better results than other queueing schemes. In the AQM group, the differences are small except for SPA that performs similarly to the other Fair Queueing schemes. The values for larger connections suffer from the bias introduced by heavy tail distributions: many very long connections do not terminate during the measurement session and those that finish benefit from shorter response times. This bias explains lower response times for connections of 6 MSS.

B. Traffic Type "Short"

In the next experiment involving Traffic Type "Short", extreme conditions (majority of very small connections) push the tested schemes to the limits with more frequent SYN losses. Figure 9 shows that **FQ CoDel, SPA, SFQ, and SYN/FIN Priority achieve the shortest response times for short connections**, while SFQ and SYN/FIN Priority are the best for the longer connections. RTT is significantly lower than in the previous experiment. This is due to the size of the transfers: with an average of 3 segments and an initial cwnd of 4 packets, most of the connections only last for the first burst of the slow start phase. Moreover, we have a proportion of 3 small segments (SYN, ACK, and FIN) for 3 data segments in the queue, virtually decreasing the queue size. Byte FIFO achieves shorter response times than AQM schemes, due to the fact that it causes almost no SYN retransmissions. The explanation of

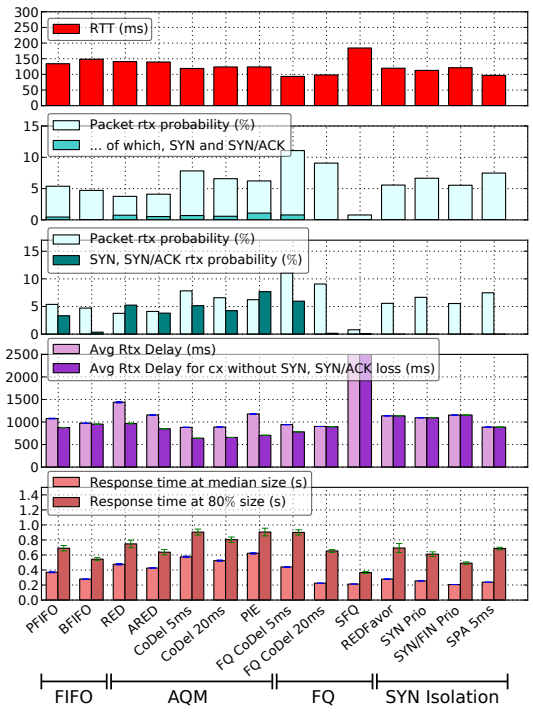


Fig. 9. Average RTT, loss percentage, retransmission, and response times under Traffic Type "Short": heavy tailed distribution with a majority of very small connections. More SYN losses than with Traffic Type "Real": larger delays (RTT and retransmission) for CoDel and PIE. FQ CoDel, SPA, SFQ, and SYN/FIN Priority display the shortest response times for short connections, SFQ and SYN/FIN Priority are the best for longer connections.

more SYN retransmissions for RED queues is inherent to their implementation in Linux: the algorithm computes the average queue length at large time intervals. When congestion appears, established connections stop sending packets, emptying the queue. Due to this interval, the algorithm takes some time to detect that the queue is again below the loss threshold and incoming packets (mostly SYN) still suffer from drops. This effect is specific to the Linux implementation of RED. In general, AQM mechanisms cause many SYN retransmissions (up to 8% of connections for PIE) that directly impact response times. The retransmission delays under SFQ stand out again due to starvation. We also note that **FQ CoDel with the default 5ms threshold delay does not perform well**. In this critical scenario, SYN packets arrive faster than the link speed. Some SYN packets stay longer in the queue than the FQ CoDel base delay and get dropped. Setting the base delay of FQ CoDel to 20 ms gives back the expected performance. SYN isolation techniques significantly improve response times compared to both their single queue counterpart and other AQM schemes. SPA also exhibits a noticeable improvement over CoDel, and performs better than a correctly configured FQ CoDel, with low RTT and retransmission delays, and some of the shortest retransmission times.

In this experiment, the results greatly differ from those for Traffic Type "Real". First of all, **delay based AQM and RED are clearly not designed for this kind of load**, with a high level of losses, and generally higher response times than with basic Packet FIFO. Byte FIFO continues to show an almost negligible amount of SYN retransmissions and remains

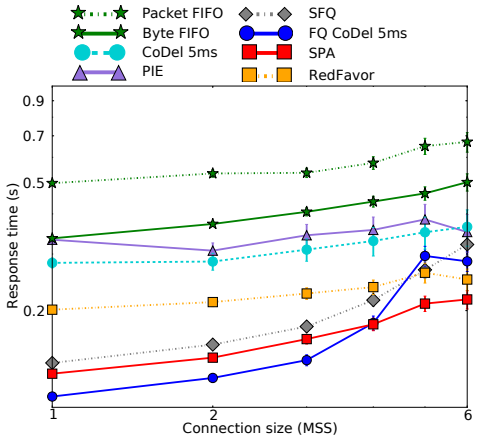


Fig. 10. Response times of connections under Traffic Type "10M" on a 10Mb/s link. FQ CoDel, SPA, and SFQ result in the best performance.

a good alternative to Packet FIFO despite a slightly lower RTT obtained by the latter. **SYN isolation techniques display a significant improvement** in terms of response times, with SYN/FIN Prio and SFQ showing the best performance. Both are closely followed by a correctly configured FQ CoDel and SPA that also obtains lower RTT and retransmission times, thus making them well suited for interactive traffic.

C. Traffic Type "10M"

We have run experiments on a 10 Mb/s link with the parameters similar to Traffic Type "Real" without reverse traffic. Figure 10 presents the results. The experiment is close to Traffic Type "Real" and the results are similar: **FQ CoDel, SPA, and SFQ perform much better than the other schemes**. Even though it is the second worst-performing scheme, Byte FIFO still obtains response times 30% lower than Packet FIFO. The main difference is that PIE and CoDel now improve performance compared to FIFO (whether packet or byte based). Moreover, REDFavor now clearly results in better performance than other AQM schemes, even outperforming SFQ and FQ CoDel for longer connections.

D. Conclusion on the Experimental Results

Based on the four setups above, we first assert that Byte FIFO is generally an interesting alternative to Packet FIFO. However, it is far more complex to implement than many other tested solutions and generally not available. Even though delay based solutions tend to compensate their high levels of SYN losses through lower RTT and faster retransmission times, they usually do not bring much improvement to the general response times and could perform a lot better with a more discerning dropping policy. Finally, isolating SYN and/or FIN in separate queues presents at least similar performance to their single-queue equivalent: in three out of four scenarios, we have observed a considerable improvement with respect to FIFO and RED based solutions. However, **SPA always performs much better than CoDel and similarly to Fair Queueing algorithms** that are drastically more complex. In some cases, it even outperforms those schemes. It does not suffer from starvation, ensuring low RTT and retransmission times in any circumstances. Moreover, it does not break down when facing

an extremely aggressive load (compared to FQ CoDel). It is, by far, the queueing scheme presenting the best performance to complexity tradeoff in our set of experiments.

VII. Conclusion

In this paper, we have evaluated the impact of SYN retransmissions on TCP connection response times based on real world traces. To achieve short response times via protecting SYN and SYN/ACK segments from losses, we have proposed SPA, a new scheme that uses two packet queues managed by CoDel—a higher priority queue for SYN and FIN segments, and a lower priority one for other segments. We have run extensive experiments on a testbed network to compare the most important schemes for three different traffic conditions. Our measurements show that the evaluated AQM mechanisms result in very similar performance: they succeed at maintaining low queueing delays although this result comes at the cost of a significant SYN loss rate. Unsurprisingly, adding some kind of Fair Queueing mechanisms consistently results in good performance, although FQ CoDel may need some tweaking in extreme traffic conditions. SFQ obtains small response times, but most losses are recovered through timeouts. However, the drawback of Fair Queueing mechanisms is complexity and a larger memory footprint compared to AQM. We also point out the fact that Byte FIFO is a simple scheme and results in very good performance. SPA, our proposal, strikes a compromise between simplicity and the memory footprint of a regular AQM policy. It achieves the low delays of CoDel and performance similar to Fair Queueing schemes. It matches high throughput and low delays required for interactive applications that cannot afford to wait for a 3 s timeout caused by a SYN loss.

Acknowledgments

This work has been partially supported by the French Ministry of Research project PERSYVAL-Lab under contract ANR-11-LABX-0025-01.

References

[1] P. Anelli, F. Harivelo, and R. Lorion, "TCP SYN Protection: An Evaluation," in *Proc. Eleventh International Conference on Networks. ICN'12*, Feb. 2012.

[2] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, pp. 20–34, May 2012.

[3] T. Hiland-Jrgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-CoDel." IETF Internet draft, March 2014. draft-hoeiland-joergensen-aqm-fq-codel-00.

[4] R. Pan, P. Natarajan, F. Baker, and G. White, "PIE: a Lightweight Control Scheme to Address the Bufferbloat Problem," Internet-Draft draft-ietf-aqm-pie-00, October 2014.

[5] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A First Look at Traffic on Smartphone," in *Internet Measurement Conference*, (New York, NY, USA), ACM, 2010.

[6] N. Kuhn, E. Lochin, and O. Mehani, "Revisiting Old Friends: is CoDel Really Achieving What RED Cannot?," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing*, ACM, Aug. 2014.

[7] M. Mellia and H. Zhang, "TCP Model for Short Lived Flows," *IEEE Communications Letters*, vol. 6, pp. 85–87, Feb. 2002.

[8] D. Ciullo, M. Mellia, and M. Meo, "Two Schemes to Reduce Latency in Short Lived TCP Flows," *IEEE Communications Letters*, vol. 13, no. 10, pp. 806–808, 2009.

[9] D. Damjanovic, P. Gschwandtner, and M. Welzl, "Why Is This Web Page Coming Up so Slow? Investigating the Loss of SYN Packets," in *Networking 2009* (D. Damjanovic, P. Gschwandtner, and M. Welzl, eds.), (Berlin, Heidelberg), pp. 895–906, 2009.

[10] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More Is Less: Reducing Latency via Redundancy ," in *Proc. 11th ACM Workshop on Hot Topics in Networks*, (New York, New York, USA), pp. 13–18, ACM Press, 2012.

[11] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression," in *Proceedings of the ACM SIGCOMM 2013 Conference*, ACM Press, Aug. 2013.

[12] "The Wonder Shaper." http://lartc.org/wondershaper/.

[13] "DD-WRT." https://www.dd-wrt.com/site/.

[14] B. Turner, "Has AT&T Wireless Data Congestion Been Self-Inflicted?." http://blogs.broughturner.com/2009/10/is-att-wireless-data-congestion-selfinflicted.html.

[15] J. Gettys, "Bufferbloat: Dark Buffers in the Internet," *IEEE Internet Computing*, vol. 15, pp. 96–96, May 2011.

[16] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling Bufferbloat in 3G/4G Networks," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, (New York, NY, USA), pp. 329–342, ACM, 2012.

[17] C. Staff, "Bufferbloat: What's Wrong with the Internet?," *Communications of the ACM*, vol. 55, no. 2, pp. 40–47, 2012. A discussion with Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys.

[18] M. Allman, "Comments on Bufferbloat," *ACM SIGCOMM Computer Communication Review*, January 2013.

[19] M. Heusse, S. A. Merritt, T. X. Brown, and A. Duda, "Two-way TCP Connections: Old Problem, New Insight," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 5–15, 2011.

[20] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, pp. 20–34, 2012.

[21] R. Pan *et al.*, "PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem," in *IEEE HPSR 2013, Taipei, Taiwan, July 8-11, 2013*, pp. 148–155, 2013.

[22] P. McKenney, "Stochastic Fairness Queueing," *Proceedings of IEEE INFOCOM*, 1990.

[23] J. Schwardmann, D. Wagner, and M. Khlewind, "Evaluation of ARED, CoDel, and PIE," in *Advances in Communication Networking*, vol. 8846, pp. 185–191, Springer, 2014.

[24] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," tech. rep., Aug. 2001.

[25] N. Khademi, D. Ros, and M. Welzl, "The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE," in *2014 Proceedings IEEE INFOCOM Workshop, Toronto, Canada*, pp. 85–90, 2014.

[26] "FreeBSD SYN Cache Source." sys/netinet/tcp_syncache.c. Accessed on: FreeBSD-9.3.

[27] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," in *Proc. INFOCOM 2003*, pp. 1742–1751 vol.3, IEEE, 2000.

[28] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 67–82, July 1997.

[29] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 303–314, Oct. 1998.

[30] "Linux Traffic Control." http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html. Accessed: 2015-03-18.

[31] "IPMT Test Suite." http://ipmt.forge.imag.fr.

[32] "OECD Broadband Portal." https://www.oecd.org/internet/broadband/oecdbroadbandportal.htm.

[33] V. Bajpai, S. J. Eravuchira, and J. Schönwälder, "Dissecting last-mile latency characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 25–34, Oct. 2017.

[34] T. Braud, M. Heusse, and A. Duda, "TCP over Large Buffers: When Adding Traffic Improves Latency," in *Proc. 26th International Teletraffic Congress (ITC)*, pp. 1–8, Sept 2014.

Policy-oriented AQM Steering

Roland Bless, Mario Hock, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
E-Mail: firstname.lastname@kit.edu

***Abstract*—Detecting and handling network congestion in the Internet has, again, become a vital area of research. The provisioning of low latency together with high throughput is of particular interest due to the current mix of applications running in the Internet. Active Queue Management (AQM) mechanisms come with the promise of reducing queuing delays. They, however, may adversely affect throughput and network utilization and have proven to be difficult to configure. More recent AQMs, such as CoDel, PIE, and GSP are easier to configure but work with a *fixed* target delay setpoint. Depending on the traffic the same setpoint value can result either in unnecessary large delays or under-utilization of the link. *Policy-oriented AQM Steering* automatically adapts the target delay setpoint to the current traffic situation, in order to fulfill a given quality-of-service policy. Such a policy consists of a *utilization goal* and an *upper delay bound*. This improves AQM performance with varying traffic situations and makes the impact of deploying an AQM predictable. A prototypical implementation of AQM Steering for GSP showed its performance advantages compared to static AQM variants at speeds of 10 Gbit/s and 1 Gbit/s.**

I. Introduction

In the last years, the reduction of latency in the Internet has become an increasingly important topic. Applications that especially benefit from low latencies are world-wide web applications due to their transactional character as well as interactive real-time applications such as Voice-over-IP or online games. However, in the current Internet such traffic is mixed with longer lasting and large data volume flows like video streams and downloads. This can cause bandwidth bottlenecks – often located at the consumer edge. As a result packets will queue up in buffers at these bottlenecks, causing an increased end-to-end delay. The "bufferbloat" studies [1] revealed that there exist several places where significantly large buffers are present and that they tend to get filled by TCP's loss-based greedy congestion control strategy. The resulting queuing delay contributes significantly to the end-to-end delay. In extreme cases, end-to-end delay can increase up to several seconds, resulting in poor TCP performance and unusable delay-sensitive applications.

An outcome of the fight against bufferbloat was to revive the use of *Active Queue Management (AQM)* in order to reduce queuing delay. The use of AQM provides several benefits, including enabling Explicit Congestion Notification [2]. Earlier efforts to deploy AQM suffered from their difficult configuration (i.e., several parameters needed to be set and their impact on performance was not obvious) and often from their negative impact on network utilization (different kinds of traffic required a different set of parameter settings to achieve a good performance). Newer AQMs such as CoDel [3], PIE [4], and GSP [5] are simpler to configure and have a *target setpoint* that corresponds to the permitted delay limit.

However, the problem of a static configuration remains since the resulting performance depends on the respective traffic. Consequently, the chosen setting can lead to sub-optimal performance [6], e.g., either a too low link utilization or a too high queuing delay. For example, CoDel uses a fixed target of 5 ms by default, which may be too low in some situations: if only a few flows traverse the bottleneck, link utilization is also low and could be increased by permitting a higher target delay. In other situations even lower targets are possible. Additionally, this means that AQMs with fixed target setpoints cannot sufficiently adapt to the current traffic situation and achieve only sub-optimal performance.

The goal of *Policy-oriented AQM Steering* is to provide an automatic adaptation, i.e., adjusting the target delay setpoint to the current traffic situation. Therefore, the AQM mechanism is controlled within given bounds that are set by a provider policy: a *lower bound for link utilization* and an *upper bound for a queuing delay target*.

AQM Steering works a on a different time-scale than AQM auto-tuning. Moreover, it is not integrated into an AQM itself. Instead, it operates as an additional control loop outside of the AQM, so that it can be easily applied to different AQMs.

II. Problem Analysis

A. Queuing Delay and Counter-Measures

As mentioned before, queuing delay often contributes substantially to the overall latency. Thus, a reduction of queuing delay (i.e., buffer occupancy) is one approach to lower end-to-end latency. Buffers in routers or switches are necessary in order to absorb short-term bursts and to keep link utilization high. There have been numerous debates about the right size of buffers in the past [7], [8]. In addition to absorbing short-term bursts, buffers have another effect on TCP performance. A large buffer allows the *congestion windows (CWnds)* of the TCP flows to inflate way beyond the *bandwidth delay product (bdp)* without causing packet losses. This creates a so-called *standing queue* within the buffer [3]. If such an inflated $CWnd$ is eventually reduced after a loss, it can still be above the bdp, thereby maintaining full link utilization. With the well-known "1-bdp Rule of Thumb" (buffer size = $1 \cdot bdp$) this is fulfilled in almost any circumstances. However, if many flows share a bottleneck *and* synchronized losses can be avoided, a smaller

ISBN 978-3-903176-08-9

$CWnd$ inflation and, thus, a smaller standing queue would suffice to keep the link fully utilized [7].

There are two different approaches to reduce the standing queue, while maintaining a high throughput:

- Use of a different congestion control that avoids to create substantial standing queues and use different backoff strategies, e.g., TCP LoLa [9] or BBR [10], which are currently under development.
- Use of *Active Queue Management* mechanisms. They try to reduce the standing queue by applying a control loop that early discards packets while retaining the buffer's capability to absorb short-term bursts. Furthermore, AQM mechanisms can support *desynchronization* of packet losses among concurrent TCP flows.

Note that using smaller tail-drop buffers is not a viable option. They lead to lower delays, but also to lower utilization: A small tail-drop buffer cannot compensate for short-term bursts and leads to synchronized packet losses. Both lead to strong backoff reactions of the TCP flows, which reduce their congestion windows way below the necessary size of $1 \cdot bdp$.

This paper focuses on Active Queue Management and assumes that currently used congestion controls, such as TCP Reno, CUBIC TCP or Compound TCP are in place.

B. Room for Improvement of Current AQMs

In contrast to earlier AQM approaches [11] newer AQMs such as CoDel, PIE, and GSP explicitly distinguish short-term bursts from standing queues and thus have a built-in burst tolerance in order to avoid unnecessary packet drops that would decrease the throughput. Moreover, earlier AQM approaches possessed several parameters that needed to be configured. The influence of the parameter setting on the achieved network utilization was often not obvious. Moreover, different traffic types required different parameter settings to achieve the best performance, i.e., they were not "self-tuning" in this respect. This turned out to be a major obstacle for their deployment [12].

Thus, newer AQMs had the objective of being usable across a wider range of scenarios without the need to adapt AQM parameters. CoDel even tries to be "parameterless for normal operation, with no knobs for operators, users, or implementers to adjust", by setting the default target value to 5 ms (5% of a 100 ms measurement interval). Nevertheless, these AQMs still have a configurable target setpoint that corresponds to the permitted delay limit. This target setpoint often relates to an internal threshold that triggers packet drops.

Even though newer AQMs are better in adapting to different traffic scenarios, they still possess their configurable but *fixed* target setpoint. This creates two-sided drawbacks, depending on the current traffic situation, which is mainly characterized by the number of *dominant* flows at the bottleneck, i.e., flows that contribute substantially to the overall in-flight data.

- *Unnecessary high delay* – In case the number of dominant flows traversing the bottleneck is large enough, the AQM can enforce its delay limit while full link utilization can be achieved, due to good loss desynchronization. However, the delay target may be excessively high (cf. Fig. 1a). It could be set to a lower value without sacrificing utilization (as in Fig. 1b).
- *Under-utilization* – In case the number of dominant flows traversing the bottleneck is low (e.g., < 10) or they are having a large RTT, the AQM cannot achieve full link utilization (see Fig. 1c). The reason is the multiplicative decrease backoff of the current TCP congestion controls. With only a few dominant flows or high RTT flows at the bottleneck, the amount of inflight data can easily fall below the bdp. In this case, a higher delay target would maintain a good link utilization (see Fig. 1d).

For the transfer of scientific data, for example, it is a typical pattern that a low number of high volume flows can appear (and disappear) as dominant flows at a bottleneck, at any time. Usually they last for a long time, e.g., hours. Thus, the traffic situation at the bottleneck is significantly changed and a bottleneck (with a fixed setpoint AQM) could fall from an unnecessary high delay to under-utilization.

The goal of Policy-oriented AQM Steering is to find the best trade-off by automatically adjusting the target setpoint within given performance bounds that are specified by a provider *policy*: a *lower bound for link utilization (u_{low})* and an *upper bound for a queuing delay target ($target_{max}$)*.

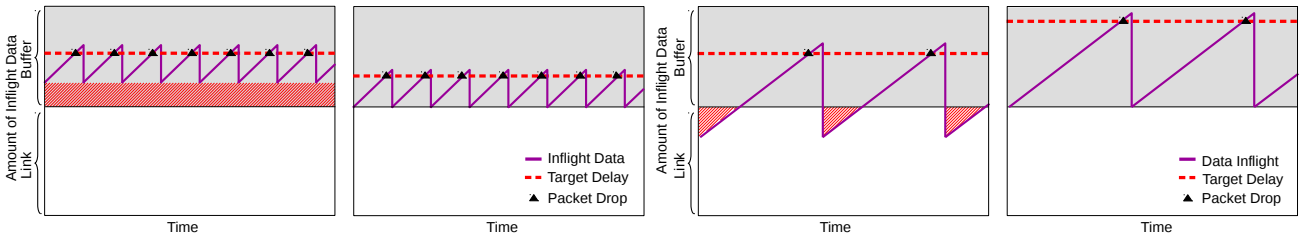
III. Design of AQM Steering

The basic principle of Policy-oriented AQM Steering is to observe how well the momentarily applied target setpoint works with the current traffic situation. If the setpoint is larger than necessary, it can be decreased without violating the lower bound for link utilization (u_{low}). If the setpoint is too small to fulfill this bound, the setpoint has to be raised, as long as the upper bound ($target_{max}$) is not reached. In order to assess the impact of the current target setpoint, the control loop of AQM Steering works outside of the AQM control loop and on a different timescale.

Fig. 2 shows the interplay of the different control loops that interact with each other. The TCP congestion control actually controls the load on the network by reacting on congestion signals (usually packet loss or ECN markings [2]). The AQM tries to find the right amount of congestion signals to emit, in order to effectively control the queue. For this, the reaction of the flows on the congestion signals has to be constantly monitored by the AQM. An important property of this interplay is that it takes at least one RTT for the congestion control to react. As soon as the AQM effectively controls the queue, AQM Steering can determine how well the target setpoint works for the current traffic situation by getting actual values for queuing delay and throughput. It also gets notified of certain events such as packet drops and then determines whether an adjustment is necessary to fulfill the given policy.

A. How can AQM Steering detect when to react?

Three different states have to be distinguished: 1) Link is no bottleneck 2) The AQM is still adjusting to the current load



(a) Target too high – unnecessary high delay (b) Adapted target – Lower delay (c) Target too low – Under-utilization (d) Adapted target – Full utilization

Fig. 1. Sketches illustrating two-sided drawbacks of AQMs with fixed target delay values

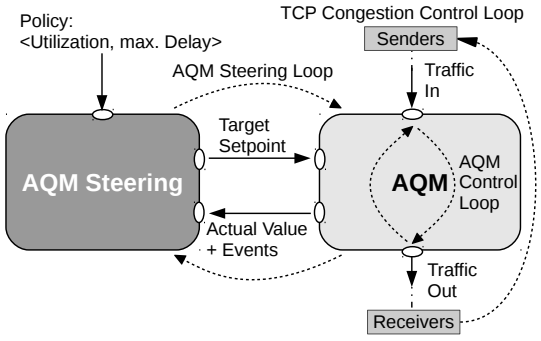


Fig. 2. Interaction of AQM Steering, AQM, and TCP Congestion Control

3) The AQM effectively controls the queue.

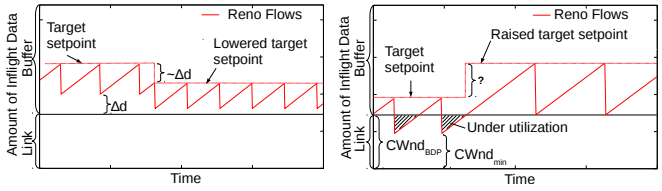
If the queue is persistently above the target, the AQM has not yet adapted to the traffic. In this case it is not expedient to change the target. After the AQM control loop has found a dropping rate that is suitable to effectively control the traffic, the queue will be fluctuating around the target. Now, AQM Steering can assess the impact of the current target setpoint. If the AQM, in contrast, does not drop any packets, the link is no bottleneck, i.e., the queue length is persistently below the target (except for bursts that are ignored by the burst protection of the AQM). Table I summarizes states, causes, and needs for action of AQM Steering.

TABLE I
AQM STEERING – STATES AND NEEDS FOR ACTION

State of Queue	Cause	Adaptation of Target
(1) persistently below target	no bottleneck, no AQM action	not necessary
(2) persistently above target	bottleneck, traffic source(s) did not respond (yet)	not useful
(3) fluctuating around target	bottleneck, AQM active	possible

B. How does AQM Steering determine a new target setpoint?

Actually, two different strategies are required: one for lowering and one for raising the target setpoint. The reason is that one can use measured queue parameters to determine how much reduction of the target is required whereas it is impossible to calculate a required increment in case of under-utilization. Both cases are sketched in Fig. 3 and will be explained in the following (the congestion window increment is sub-linear in reality due to the increasing queuing delay).



(a) Lowering the target setpoint (b) Raising the target setpoint

Fig. 3. Target adaptation problems

1) Lowering the Target Setpoint: If the target setpoint is too high, as shown in Fig. 3a, a persistent minimum standing queue can be measured that does not dissipate even after packet drops. Thus, AQM Steering can determine the minimal queuing delay Δd and reduce the target setpoint by approximately this amount. The key point behind this strategy is that the reaction of TCP flows on a congestion signal depends only marginally on the actual value of the target setpoint. This means that even with the lower setpoint, the queue will not underflow after a congestion signal. Thus, the queuing delay is reduced without harming throughput.

However, if multiple flows are present at a bottleneck, their $CWnd_i$ differ in size and the effect of the individual backoff of each flow might be different. A single Δd as sketched in the simplified exemplary situation shown in Fig. 3a is therefore not sufficient: Δd will actually vary. Thus, an average $\overline{\Delta d}$ of measured minima Δd_i and their variance (σ_n) are calculated. The new target setpoint is calculated as follows:

$$target_{new} = \min\left(target_{old} - \overline{\Delta d} + \gamma\sigma_n, target_{old}\right),$$

with $\sigma_n = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\Delta d_i - \overline{\Delta d})^2}$. $\gamma > 1$ defines a protective margin so that an adaptation does not occur too often if the fluctuation is high. A smaller γ leads to a smaller target value and thus more likely to under-utilization.

2) Raising the Target Setpoint: If the target setpoint is too low to achieve full utilization, it is impossible to calculate the required increment for the target setpoint, because there is no directly usable correlation without knowledge of the flow's bdp, that depends on its RTT. Further dependencies are the backoff factor β (which is different for CUBIC TCP and TCP Reno) and the number of flows. This knowledge is

not typically available at the intermediate system that operates the AQM. Therefore, an approach for a stepwise increment is used.

In order to quickly restore a reasonable throughput when under-utilization has been detected, the target setpoint is restored to a reasonable default value ($thresh_{min,up}$). Since one goal of the AQMs that use a fixed setpoint is to provide a reasonable default value, we recommend to use their default value as $thresh_{min,up}$. If the under-utilization persists, the setpoint is multiplicatively increased (e.g., by a factor 2) until the lower utilization bound u_{low} is fulfilled or the upper limit for the setpoint $target_{max}$ is reached:

$$target_{new} = \min\Big(\max\left(target_{old} \cdot 2, thresh_{min,up}\right), target_{max}\Big)$$

The adaptation happens only if a *short-term smoothed value* as well as a *longer-term smoothed value* of the measured data rate is below the lower utilization bound u_{low}. The short-term smoothed value is simply the mean data rate measured since the last packet drop. This value then contributes to a time-dependent longer-term smoothed value. Further details are given in Sec. IV.

C. Policy Option: Under-utilization

Policy-oriented AQM Steering also allows to define an upper bound u_{target} on the utilization with $u_{low} \leq u_{target} \leq 100\%$. The advantage of using u_{target} is that queuing delay is avoided while there is still a lot of room for short-lived flows as well as other more bursty short-lived traffic. Some providers, for example, avoid utilizations above 50% in order to provide enough failure protection capacity.

To achieve this goal it may be necessary to discard packets earlier, even before any packets queue up. To achieve this, AQM Steering can optionally switch the AQM from the physical queue to a virtual queue. For this a virtual egress rate $rate_{virtual}$ (with $rate_{virtual} < rate_{physical}$) is defined. The virtual queue tracks the notional buffer utilization that would have built up if the egress link had a data rate of $rate_{virtual}$. The switch to the virtual queue is performed as soon as the target setpoint has reached the minimum value and the current utilization u is still larger than u_{target}. This way, the burst tolerance and loss desynchronization provided by an AQM can still be used, even without any physical queue.

It has to be noted that the virtual queue is just a passively computed number, i.e., the traffic is *not* shaped to the virtual egress rate, since this would induce real queuing delay. Therefore, the link utilization can be above $rate_{virtual}$ for short periods of time. Still, the average link utilization will be below or equal to $rate_{virtual}$. Otherwise the virtual queue would increase indefinitely, which is prevented by the AQM. Whether the average link utilization is actually equal to $rate_{virtual}$ depends on the traffic and the level of loss synchronization. This means that the AQM Steering control loop is still necessary to keep $u_{low} \leq u \leq u_{target}$. However, on the virtual queue AQM Steering does not control the target setpoint. Instead, it adjusts $rate_{virtual}$ in the range of $u_{target} \cdot rate_{physical} \leq rate_{virtual} < rate_{physical}$. If $rate_{virtual} = rate_{physical}$ is necessary to avoid $u < u_{low}$, the AQM is seamlessly switched back to the physical queue. More details are provided in Sec. IV.

D. Discussion

AQM Steering cannot always achieve u_{low}. If the link is no bottleneck, it will inevitably be underutilized. Also, a larger standing queue than allowed by $target_{max}$ might be necessary to attain u_{low}. Hence, $target_{max}$ can be understood as maximal delay one is willing to trade for higher throughput. Often large tail-drop buffers are deployed to get a high throughput at the cost of delay. When converged, the link utilization of AQM Steering will be at least $\min\Big(u_{low}, thr_{td}(target_{max})\Big)$, with $thr_{td}(\ldots)$ being the throughput of a tail-drop buffer of the given size would achieve with the same traffic. Due to the burst protection and loss desynchronization provided by an AQM, the throughput of AQM Steering can actually be larger than $thr_{td}(target_{max})$.

During the convergence of the AQM Steering control loop, throughput can be below $thr_{td}(target_{max})$. Therefore, AQM Steering is tuned to quickly converge towards larger target setpoints (i.e., improving link utilization). Reducing the setpoint (i.e., lowering delay) is performed more conservatively.

IV. Implementation

We chose to use the GSP AQM [13], [5] as basis for our prototype implementation as it is simple to implement and achieves comparable results to CoDel and PIE. GSP's target setpoint is the packet queuing delay in form of a *threshold*. If the *threshold* is exceeded (the sojourn time of the last dequeued packet was larger than threshold) on packet arrival, it immediately discards the arriving packet. GSP then pauses discarding for a time span (called *interval*) allow the congestion control to react on the drop. The *interval* is adapted according to the situation, i.e., if the congestion control reaction was not effective enough to let the queuing delay fall below the threshold, the interval becomes shorter, so that GSP drops more aggressively.

Due to performance and simplicity reasons, AQM Steering was integrated into the GSP implementation, although the general concept allows for a more modular and separated realization. The threshold adaptation of *GSP with AQM Steering (GSP-AS)* is hooked-in into the threshold exceeded event, i.e., $t_{sojourn} > thresh_{curr}$ but is carried out before a packet drop is performed. This way, an increase of the threshold will defer a packet drop until the increased threshold is exceeded. If the threshold is kept unchanged or lowered, a packet is dropped as usual.

Table II shows different variables that are also used in the following description. Policy-oriented AQM Steering allows to set $thresh_{max}$ as well as $rate_{target,min}$ as parameters. Optionally, a third parameter $rate_{target}$ can be set if the desired operational mode is under-utilization (see Sec. III-C). The short-term link utilization is calculated by $rate_{bd} =$

TABLE II
SELECTED VARIABLES

Name	Default	Explanation
$thresh_{max}$ *)	configurable	upper bound for the target setpoint
$thresh_{min}$	0,2 ms	lower bound for the target setpoint
$rate_{bd}$	–	data rate since last packet discard
$rate_{bd,avg}$	–	longer-term smoothed data rate
$rate_{max}$	–	maximum link speed, corresponds to $u = 100\%$
$rate_{target,min}$ *)	configurable	lower bound on rate, corresponds to u_{low}
$rate_{target}$ *)	optionally configurable	rate that corresponds to target utilization u_{target}

*) policy is expressed by these parameters

$\sum_{i=0}^{n} packet_{t_i}.size/(t - t_0)$, where t_0 is the time of last packet discard, $packet_{t_i}$ denotes a packet that arrived at time $t_i \in [t_0, t]$. Based on these values the longer-term $rate_{bd,avg}$ is calculated with the TDRM-UTEMA-CPA smoothing function [14].

Algorithm 1 Lowering the Target Setpoint

```
1: procedure ATGSPINTERVALADAPTATION
2:  ...
3:  t_rq_min ← min(t_current_sojourn, t_rq_min)
4:  ...
5: V, V_avg ← 0                                   ▷ Initialize at start
6: thresh_curr, t_rq_min ← [thresh_min, thresh_max]
7: procedure ATGSPPACKETDISCARD
8:  if t_rq_min < thresh_curr then
9:   Δt_rq_min ← thresh_curr − t_rq_min
10:  Δt_rq_min_avg ←SMOOTHUTEMA(Δt_rq_min, now)
11:  ESTIMATEVARIANCE(Δt_rq_min)
12:  thresh_down ← thresh_curr − (Δt_rq_min_avg + γ√V_avg)
13:  thresh_down ← ⌊thresh_down/thresh_down_min⌋ · thresh_down_min
14:  if thresh_down > 0 and rate_bd,avg > rate_target,min then
15:   thresh_curr ← max(thresh_curr − thresh_down, thresh_min)
16:  t_rq_min ← thresh_curr
17:  ...
```

Algorithm 1 shows the pseudocode for lowering the target setpoint. As discussed above, the target setpoint should not be adjusted if the queue is persistently above the target (see Table I). Therefore, the minimum packet sojourn time t_{rq_min} (rq indicates the real queue, vq the virtual queue) that occurs between two packet discards is tracked and the adaptation is only performed if $t_{rq\_min} < thresh_{curr}$ (line 8). This can be done when the GSP interval is adapted anyway (see line 3).

As visualized in Fig. 3, the adaptation amount is calculated based on $\Delta t_{rq\_min} = thresh_{curr} - t_{rq\_min}$. This value is smoothed with the UTEMA function [14] and a variance is calculated. The threshold decrement gets rounded to an integral multiple of $thresh_{down\_min}$ (the minimal allowed threshold value, e.g., 0.2 ms) to avoid too small adjustments (see line 13). This also increases the stability for very low thresholds in combination with smoothing of the measured values. To improve the stability of the mechanism, the new threshold is only applied if the longer-term measurement for the data rate $rate_{bd,avg}$ is above the configured lower bound rate $rate_{target,min}$ (line 15).

If a target utilization goal u_{target} is specified, the decrement by $thresh_{down}$ may not be sufficient to get there. Therefore, a step-wise multiplicative reduction is applied until the minimum threshold $thresh_{min}$ is reached (shown in algorithm 3). If a further reduction is required, the AQM Steering switches to operate on the virtual queue.

Algorithm 2 shows the pseudocode for raising the target. A precondition is that the queue has been drained empty after a packet drop (line 3). If the link is no bottleneck, raising the threshold will not increase the utilization. If a drop will not cause the buffer to empty, link utilization already is at 100 %. Thus, increasing the threshold is not necessary. If line 3 is true, the short-term $rate_{bd}$ *and* the longer-term $rate_{bd,avg}$ are checked against the configured minimum $rate_{target,min}$ (u_{low}). If both are below this target, the threshold is raised (line 10). Otherwise, $rate_{bd} > rate_{target,min}$ alone would often trigger too early, whereas $rate_{bd,avg} > rate_{target,min}$ alone could lead to an unnecessary raise, because of the inertia of $rate_{bd,avg}$, i.e., the necessary threshold could have been reached already in the meantime. If the increase conditions are met, the threshold is multiplicatively increased by a factor $\alpha = 2$. However, if the threshold is very low, the reaction on a sudden load change could be too slow. Therefore, the threshold is at least set to $thresh_{min,up} := 0.025 interval_{init}$. We decided to set this value similar to the default GSP parametrization. This way, the throughput with GSP-AS will be at least as high as with regular GSP after an underutilization has been detected.

Algorithm 2 Raising the Setpoint Target

```
1: procedure ATGSPPACKETARRIVAL
2:  ...
3:  if (queue empty) and (packet was discarded) then
4:   thresh_up ← true
5:  if t_sojourn > thresh_curr and now > timeout_expiry then
6:   rate_bd ← Data rate since last packet discard
7:   if thresh_up = true then
8:    if rate_bd < rate_target,min and rate_bd,avg < rate_target,min then
9:     thresh_curr ← min(max(thresh_curr · α,
10:                          thresh_min,up), thresh_max)
11:   thresh_up ← false
12:  else
13:   rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
14:   GSPPACKETDISCARD
15:  ...
```

Algorithm 3 shows the switch to the virtual queue in case the configured target $rate_{target}$ has not been reached by lowering the threshold as shown in algorithm 1. However, the threshold adaptation cannot be applied in the same way as for the real queue, because the virtual queue does not effectively delay packets. In order to maintain the AQM's property of achieving desynchronization, the virtual departure rate $rate_{vq}$ is adapted instead (line 12). Since the flows congestion windows will fluctuate, $rate_{vq}$ will be often higher than $rate_{target}$, but leading to an effective measured $rate_{target}$ due to the oscillations. Therefore, $rate_{vq} \in [rate_{target}, rate_{max}]$. If the average rate $rate_{bd,avg}$ is lower than the target $rate_{target}$

then $rate_{vq}$ will be raised. If $rate_{vq}$ reaches $rate_{max}$ then the operation uses the real queue again. This is shown in algorithm 4.

Algorithm 3 Rate adaptation and switch to virtual queue

```
1: boolean vq_active                                   ▷ Packet discard according to virtual queue
2: procedure ATGSPPACKETDISCARD
3: ...
4:  rate_bd ← Data rate since last packet discard
5:  rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
6:  if rate_bd,avg > rate_target and rate_target < rate_max then
7:   if vq_active = false and rate_bd > rate_target then
8:    thresh_curr ← max(thresh_min, thresh_vq, thresh_curr · 0.75)
9:    if thresh_curr = thresh_vq then
10:     vq_active ← true
11:  if vq_active = true then
12:   rate_vq ← max(rate_target, rate_vq + α(rate_target − rate_bd,avg))
13: ...
```

Algorithm 4 Rate adaptation and change to real queue

```
1: boolean vq_active                                   ▷ Packet discard according to virtual queue
2: procedure ATGSPPACKETARRIVAL
3: ...
4:  if vq_size > thresh_vq and now > timeout_expiry  then
5:   rate_bd ← Data rate since last packet discard
6:   if thresh_up = true then
7:    if rate_bd < rate_target,min and rate_bd,avg < rate_target,min then
8:     rate_vq ← min(rate_max, rate_vq + α(rate_target − rate_bd,avg))
9:     if rate_vq = rate_max then
10:     vq_active ← false
11:   thresh_up ← false
12:  else
13:   rate_bd,avg ← SMOOTHTDRM(rate_bd, now)
14:   GSPPACKETDISCARD
15: ...
```

V. Evaluation

We evaluated Policy-oriented AQM Steering at speeds of 1 Gbit/s and 10 Gbit/s. One objective was to compare the performance of AQM Steering with tail-drop buffers (small and large) and AQM approaches with fixed targets (namely CoDel and GSP). Another objective was to evaluate the adaptivity of AQM Steering with respect to changing traffic situations. The following experiments were performed: Steady state with long-lived flows, steady state with long-lived and short-lived flows, and transition behavior for changing traffic situations. An additional experiment (with $u_{target} = 95\%$) was performed to evaluate the virtual queue feature. Every experiment was repeated in 10 different runs. Due to space restrictions we present only the results of the 10 Gbit/s experiments, the results for the 1 Gbit/s experiments are very similar.

A. Testbed

Fig. 4 shows the configuration of the testbed where the experiments were conducted. Sender and DPDK-based switch were dual processor servers equipped with Intel Xeon E5-2630, the receiver was a dual processor Intel Xeon E5-2640. All servers were equipped with 128 Gbyte RAM and Intel X710 4-port 10 Gbit/s network cards. The Ubuntu 16.04 Linux distribution was used as operating system, Kernel versions 4.9.0 and 4.4.0 were used. Generic Receive Offload and Generic Segmentation Offload were active. The DPDK switch used bursts of 32 packets and had a configured limit for the send and receive ring buffers of 256 packets.

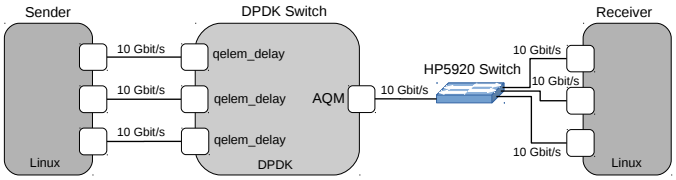


Fig. 4. The Testbed in its 10 Gbit/s Configuration

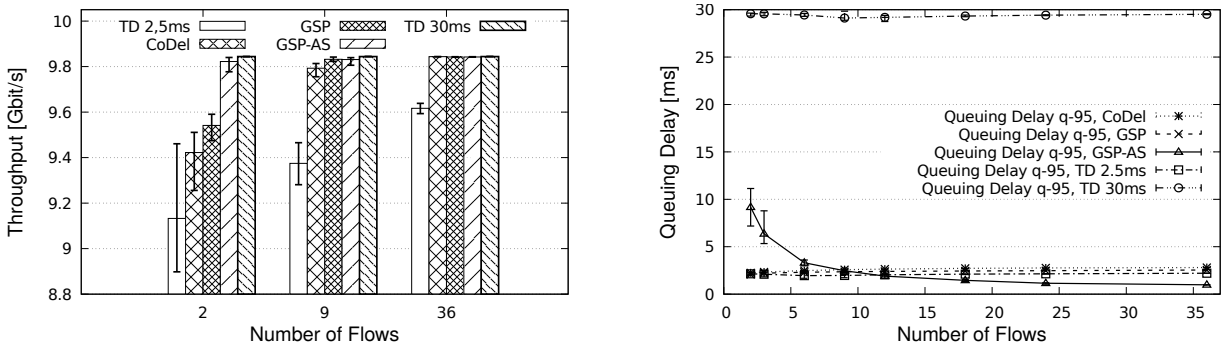
The DPDK-based switch[1] implemented the AQMs CoDel, GSP as well as GSP-AS. It also allows for monitoring internal AQM state such as packet drops and queue length. Moreover, the module `qelem_delay` generates artificial delay, since `netem` is not able to generate reliable behavior at such high speeds. The RTT for all experiments was set to 50 ms. As tools `TCPlog`[2], `CPUnetLOG`[3], and `iperf3` were used.

B. Steady State – Long-lived Flows

This experiment shows that AQM Steering adapts to the traffic situation and achieves higher throughput at the lowest possible queuing delay in comparison to static AQMs or simple tail-drop buffers. Fig. 5 shows GSP-AS in comparison with statically configured GSP and CoDel as well as small (2.5 ms) and large (30 ms) tail-drop (TD) buffers. Static GSP and CoDel were used with target setpoints of 2.5 ms. The upper delay limit $thresh_{max}$ was set to 30 ms, u_{target} was set to 100%, and u_{low} to 99%. CUBIC TCP flows were used (with their standard $\beta_{Cubic} = 0.7$) as traffic load and the number of flows was varied as follows: 2, 3, 6, 9, 12, 18, 24, 36. For clarity Fig. 5a shows only results for 2, 9, and 36 flows.

As expected, the throughput is lower for static AQMs and the small tail-drop buffer if the number of flows is low (in this setting, < 9). This shows that GSP-AS increases the delay in order to achieve higher throughput as desired by the policy. In contrast to the tail-drop buffer, it only increases the delay as necessary, i.e., GSP-AS stays clearly below the allowed 30 ms, as shown in Fig. 5b. This as well as other plots show the average of the 95%-quantile across the 10 runs, the error bars their respective min/max values. The curve for GSP-AS show that if the number of flows becomes higher, GSP-AS can reduce queuing delay even further by lowering the target setpoint. While the static variants of GSP and CoDel also accomplish good throughput values, because of the achieved desynchronization, they stay at their fixed targets of 2.5 ms, whereas GSP-AS can go as low as 0.825 ms.

[1] https://git.scc.kit.edu/TM/DPDK_AQM_Switch (branch: AQM_Steering)
[2] https://git.scc.kit.edu/CPUnetLOG/TCPlog
[3] https://git.scc.kit.edu/CPUnetLOG/CPUnetLOG



(a) Throughput

(b) Queuing Delay

Fig. 5. Throughput and Queuing Delay for Different Numbers of Long-Lived Flows

C. Steady State – Long-lived and Short-lived Flows

This experiment shows that AQM Steering still works if short-lived flows disturb the AQM control: static AQMs loose throughput whereas AQM Steering achieves high throughput at lowest possible queuing delays. Several experiments were made where long-lived TCP flows are disturbed by short-lived TCP transfers of 64 MByte every two seconds. The short-lived flows are usually finished before their slow start phase ends. Although AQMs like CoDel, PIE, and GSP have a kind of burst protection already, short-lived transfers may still decrease the throughput significantly. Fig. 6a shows that static AQM variants perform much worse with a lower number of flows (cf. Fig. 5a). Bursty traffic causes packet drops and it is likely that long-lived flows are affected. The queue within the small tail-drop buffer drains completely between drops as can be seen by the very low throughput values and the practically non-existing delay (Fig. 6b). GSP-AS is able to keep the throughput high (even higher than TD 30 ms), at the cost of increasing the queuing delay. With 36 flows GSP-AS is able to reduce the delay without hurting the throughput.

D. Transition Behavior

This experiment shows how AQM Steering adapts when a sudden change of the traffic situations happens. In contrast to the previous experiments, the traffic situation (here the number of flows) changes during the experiment. At first, only two data flows are started and get to steady state when at $t = 180\,s$ 34 additional flows are started that last for 80 s. Fig. 7a shows how AQM Steering smoothly lowers the target setpoint as the number of flows is higher and raises the target setpoint relatively quickly after the remaining two flows have raised their $CWnd$ up beyond the bdp (that takes 18 s) after the 34 flows have ended at second 260. Conceptually, AQM Steering cannot compensate the time TCP requires to claim free bandwidth after other flows have finished, therefore, the target setpoint is not adapted in absence of any packet drops.

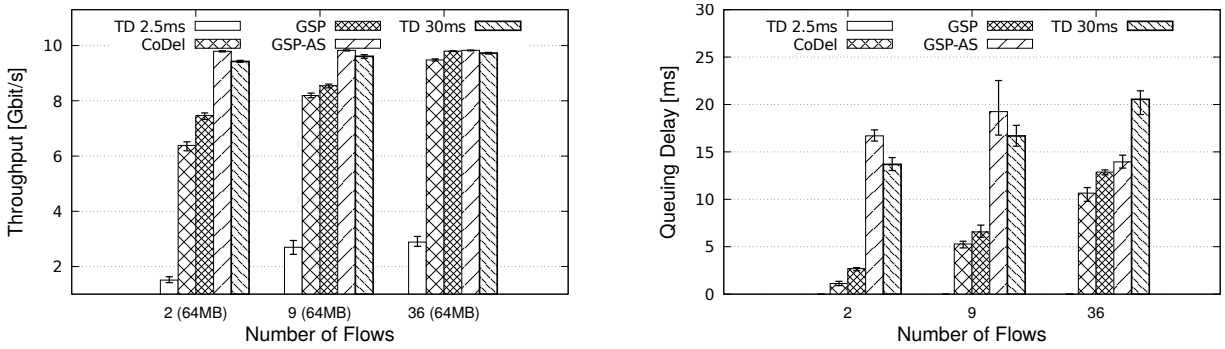
Fig. 7b shows results of an experiment where the situation is repetitively changed before the adaptation of AQM Steering has converged. Every 40 s, 34 additional flows arrive that disappear again after 20 s. During convergence AQM Steering favors high link utilization over a quick reduction of the queuing delay. Indeed, the threshold almost reaches the same upper values as it settled on in the previous experiment. Reduction of the target setpoint, in contrast, is more conservative.

E. Policy Option: Under-utilization

To evaluate the virtual queue feature some experiments were performed with $u_{target} = 95\%$ and $u_{low} = 94\%$. The generated traffic is the same as in Sec. V-B. Fig. 8a shows that the achieved throughput stays within the utilization bounds $[0.94, 0.95]$ (see upper curve and right y-axis). Furthermore, starting at six flows the real queue length is effectively zero (as shown by the queue length curves of the average and the 95% quantile), since the AQM triggers regularly packet discards based on the virtual queue. For a lower number of flows, the virtual queue length fluctuates more (as explained in Sec. II), thus often overshooting into the real queue, as also indicated by the difference of the 95% quantile and the average queue length. Fig. 8b shows the virtual queue length and packet discard actions from a run with 36 concurrent flows. As with the physical queue, the virtual queue length fluctuates around the target. Since the virtual queue is almost never empty, the egress rate is close to $rate_{virtual}$, on average. AQM Steering adjusts $rate_{virtual}$ in order to keep $u_{low} \leq u \leq u_{target}$.

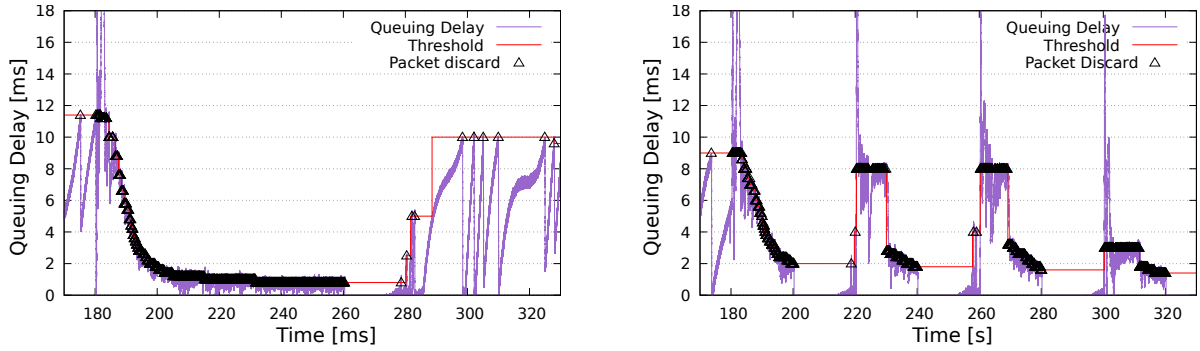
VI. Related Work

The first major AQM mechanism RED faced severe deployment difficulties, due to its sensitivity to parameters, which were hard to tune. Since then auto-tuning has become a standard functionality of subsequent AQMs, like Adaptive RED (ARED) [15], BLUE [16], Optimal Drop-Tail/Optimal BLUE [17]. The latter approach tries to optimally trade-off delay versus throughput based on utility functions. [11] gives an exhaustive survey of existing AQMs. Newer AQMs like CoDel [3], PIE [4], and GSP [5] try to be mostly parameterless, due to inherent auto-tuning and reasonable default values. But due to their fixed delay target setpoint, the performance still depends on the traffic characteristics and is, therefore, hard to predict. Policy-oriented AQM Steering focuses on an automatic adjustment of this parameter, in order give better and more predictable performance.



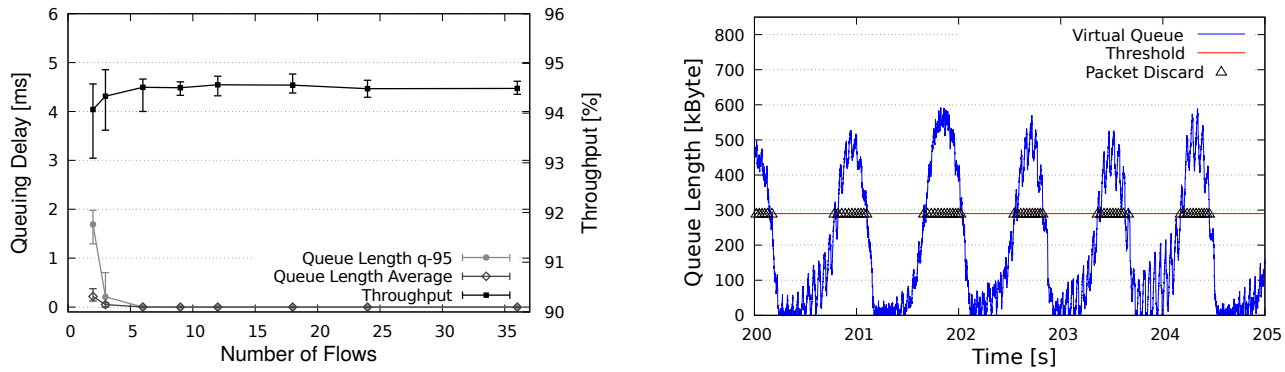
(a) Throughput (b) Queuing Delay

Fig. 6. Throughput and Queuing Delay for Different Numbers of Long-Lived Flows Disturbed by Short Flows (64 MByte)



(a) Slow changes (b) Faster changes

Fig. 7. Transition Behavior



(a) Achieved Throughput and Queuing Delays (b) Length of the Virtual Queue (36 flows)

Fig. 8. Behavior for $u_{target} = 95\%, u_{low} = 94\%$ and upper delay limit 2.5 ms

The work in [6] assesses the operating ranges and the tunability of the AQMs CoDel and PIE. The authors conclude that "manual tuning can hardly be avoided" for some use-cases that lie outside the operating range of the default parameter set. But even for scenarios within the operating ranges, different trade-offs between queuing delay and throughput are possible.

These trade-offs can be tuned by altering the target setpoint. Policy-oriented AQM Steering does exactly this but in an automatic manner. The authors of [6] also found that adapting the update interval λ to the actual RTTs might be useful in some cases. Our mechanism does not tune this parameter, since the RTTs are usually not known by routers/switches. Furthermore, GSP (which our implementation is based upon) does not have such a fixed update interval as CoDel and PIE.

The concept of virtual queues was first introduced as part of [18]. Based on this concept the AQMs AVQ [19] and HULL [20] have been developed. HULL uses so-called phantom queues that simulate queue buildup for a virtual egress link that runs at a fixed fraction of the actual link (e.g., 95 %), with the goal to leave "bandwidth headroom". HULL is designed to be used in conjunction with DCTCP [21]. AVQ simulates a virtual tail-drop queue with a variable virtual

egress rate. The virtual rate is adjusted according to the length of the physical queue, in order to achieve a certain ingress rate ($\leq 100\%$). Our approach also uses a virtual queue, if an upper utilization target is set that cannot be achieved by operating on the physical queue. It further differs from the other approaches by using an AQM in order to achieve a good loss desynchronization in combination with an outer control loop that regulates the virtual egress rate.

VII. Conclusion

Policy-oriented AQM Steering provides an external control loop that dynamically adjusts the target setpoint of newer AQMs. Depending on the traffic this can lead either to lower queuing delays or higher utilization of the bottleneck link. Without AQM Steering, AQMs provide a trade-off between link utilization and delay that is hard to determine, since it changes under different traffic situations. With AQM Steering a simple to grasp policy can be set, consisting of: $\langle u_{low}, target_{max} \rangle$ and optionally u_{target}. This makes the deployment of AQM more predictable and can even improve the performance, e.g., if traffic patterns change over time or are different than expected.

As shown in the evaluation, AQMs can cause a significant drop in link utilization (down to 60 %–80 %) under certain circumstances. Network providers could, therefore, be reluctant to deploy AQMs. In these cases, higher link utilization can be attained at the cost of permitting a larger queuing delay. AQM Steering's policy allows network providers to specify how much queuing delay they are willing to trade for high throughput. But in contrast to large tail-drop buffers (or statically configured AQMs with high target setpoints), AQM Steering only permits these delays when necessary. Otherwise, the delay is reduced to the minimal value that is required to achieve the desired throughput. In addition to that, AQM Steering can optionally switch the AQM to a virtual queue, which allows to specify upper utilization targets. This enables policies that focus on zero queuing delay by enforcing spare capacity. The evaluation has shown that the concept works well for different traffic situations. When traffic patterns change, AQM Steering requires some time to adapt. But due short-term and longer-term smoothing, quickly changing traffic situations do not destabilize the control. Investigation of AQM Steering in more complex scenarios and with different traffic mixes is planned as future work.

Acknowledgment

The authors would like to thank Moritz Kunze for his contributions, thorough and intensive work on the topic, implementation, and evaluation. This work was supported by the bwNET100G+ project, which is funded by the Ministry of Science, Research, and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

References

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[2] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–63, Sep. 2001.

[3] K. Nichols and V. Jacobson, "Controlling Queue Delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[4] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, July 2013, pp. 148–155.

[5] W. Lautenschlaeger and A. Francini, "Global Synchronization Protection for Bandwidth Sharing TCP Flows in High-Speed Links," in *Proceedings of 16th International Conference on High Performance Switching and Routing (IEEE HPSR 2015)*, Jul. 2015, budapest, Hungary.

[6] N. Kuhn, D. Ros, A. B. Bagayoko, C. Kulatunga, G. Fairhurst, and N. Khademi, "Operating ranges, tunability and performance of CoDel and PIE," *Computer Communications*, vol. 103, no. Supplement C, pp. 74–82, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366416302717

[7] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1030194.1015499

[8] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on Router Buffer Sizing: Recent Results and Open Problems," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 34–39, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1517480.1517487

[9] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct 2017, pp. 215–.218.

[10] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[11] R. Adams, "Active Queue Management: A Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1425–1476, Q3 2013.

[12] F. Baker (Ed.) and G. Fairhurst (Ed.), "IETF Recommendations Regarding Active Queue Management," RFC 7567 (Best Current Practice), RFC Editor, Fremont, CA, USA, pp. 1–31, Jul. 2015.

[13] W. Lautenschlaeger, "Global Synchronization Protection for Packet Queues," Internet-Draft draft-lauten-aqm-gsp-03, May 2016, work in progress, https://tools.ietf.org/html/draft-lauten-aqm-gsp-03.

[14] M. Menth and F. Hauser, "On Moving Averages, Histograms, and Time-Dependent Rates for Online Measurement," *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, Apr. 2017, preprint. [Online]. Available: https://atlas.informatik.uni-tuebingen.de/~menth/papers/Menth17c.pdf

[15] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," AT&T Center for Internet Research at ICSI, Tech. Rep., 2001.

[16] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The BLUE Active Queue Management Algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, Aug 2002.

[17] R. Stanojević and R. Shorten, "Trading link utilization for queueing delays: An adaptive approach," *Computer Communications*, vol. 33, no. 9, pp. 1108–1121, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366410000897

[18] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, no. 12, pp. 1969–1985, 1999.

[19] S. S. Kunniyur and R. Srikant, "An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 286–299, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1109/TNET.2004.826291

[20] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228324

[21] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851192

Adaptive Robust Traffic Engineering in Software Defined Networks

Davide Sanvito, Ilario Filippini, Antonio Capone
Politecnico di Milano
name.surname@polimi.it

Stefano Paris, Jeremie Leguay
France Research Center, Huawei Technologies Co. Ltd
name.surname@huawei.com

***Abstract*—One of the key advantages of Software-Defined Networks (SDN) is the opportunity to integrate traffic engineering modules able to optimize network configuration according to traffic. Ideally, the network should be dynamically reconfigured as traffic evolves, so as to achieve remarkable gains in the efficient use of resources with respect to traditional static approaches. Unfortunately, reconfigurations cannot be too frequent due to a number of reasons related to route stability, forwarding rules instantiation, individual flows dynamics, traffic monitoring overhead, etc.**

In this paper, we focus on the fundamental problem of deciding whether, when and how to reconfigure the network during traffic evolution. We propose a new approach to cluster relevant points in the multi-dimensional traffic space taking into account similarities in optimal routing and not only in traffic values. Moreover, to provide more flexibility to the decisions on when to apply a reconfiguration, we allow some overlap between clusters that can guarantee a good-quality routing regardless of the transition instant.

We compare our algorithm with state-of-the-art approaches in realistic network scenarios. Results show that our method significantly reduces the number of reconfigurations with a negligible deviation of the network performance with respect to the continuous update of the network configuration.

I. Introduction

Traffic Engineering (TE) [1] plays a crucial role for service providers since it permits to optimize network performance, reduce operational costs, and load balance the utilization of network resources. However, the dynamic nature of the traffic due to ordinary daily fluctuations and unpredictable events stirs up the trade-off between optimality of the routing configuration and network reconfiguration rate. The traditional approach of service providers is to optimize the routing considering the "worst case" traffic condition so as to rarely reconfigure the network. The resulting overprovisioning leads to the underutilization of network resources.

Software-Defined Networks (SDNs) [2] provide the needed flexibility to update more frequently TE policies. Having a global view of the network status, SDN controllers can integrate TE algorithms [3]–[5] to continuously optimize the network with an online twist. As the system evolves, new configurations are applied to the network equipment to optimize network performance. However, sudden and unpredictable system changes, like traffic variations, network failures, and the uncontrolled rate of change, still pose a major challenge for these methods.

ISBN 978-3-903176-08-9

The solutions that have been devised to cope with traffic variations can be broadly classified into three main classes: *dynamic TE*, *static TE*, and *semi-static TE*. While different in the way they calculate network configurations, all techniques require the use of Traffic Matrices (TMs) periodically collected by a network monitoring tool. *Dynamic TE*, like [3], [4], [6], [7], uses such information to predict the next system state and compute the corresponding optimal routing configuration using linear programming [8] or fast approximation algorithms [9]. The accuracy of the prediction highly affects the optimality of the computed solution while frequent network reconfigurations result in control plane congestion due to the low speed of flow programming [4] in hardware. In contrast, *static TE*, such as *oblivious routing* [10] and robust routing [11]–[13], monitors TMs over a long period of time and computes the TE configuration that minimizes the worst deviation with respect to the sequence of all optimal configurations. This class of TE policies keeps the network configuration stable, but it inevitably suffers from low optimality during most of the operational time.

Semi-static TE approaches such as [14], [15] combine both static and dynamic TE to approximate the optimal sequence of configurations with a limited set of routing solutions computed over clusters of TMs. Clusters of TMs are formed either by statically dividing time in different intervals or by finding similarities in the traffic domain. However, the arbitrary splitting of the time domain results in significant performance loss when sharp traffic variations are temporally close. Similarly, using the same routing configuration for TMs that are close in the traffic domain (i.e., their entries have the same magnitude) but far in the time domain can lead to frequent network reconfigurations. More importantly, the controller needs to decide whether and when to reconfigure. Transitory traffic fluctuations should be ignored to avoid system oscillations and the network should be reconfigured only when it is evolving towards a new state.

In this paper, we study the fundamental problem faced by SDN controllers of deciding whether, when and how to reconfigure the network after a traffic evolution. To provide an answer we study and address the problem of building a set of robust routing configurations associated to clusters of TMs that overlaps in time, traffic and routing domains. Time overlap refers to the amount of time we are able to use a routing configuration even for TMs that are outside the associated

cluster with minimal efficiency degradation. Traffic overlap denotes the similarity in the traffic space of TMs within the same cluster, whereas routing overlap indicates how similar routing configurations associated to two different clusters are.

Given the interplay between TM clusters and routing solutions, we decouple the problem into two subproblems, namely TM clustering and robust routing. To this aim, we propose Clustered Robust Routing (CRR), an iterative algorithm that achieves three objectives: 1) covering the entire TM space so that a feasible routing configuration is available for any traffic condition, 2) reducing the number of routing changes by creating a small set of robust routing configurations that can be used for a minimum duration each time one of them is applied, and 3) maintaining a minimum time overlap between adjacent clusters that can be exploited to decide whether to reconfigure the network. We analyze our algorithm on a realistic network scenario and compare its performance against state of the art approaches of the three TE classes discussed above.

This paper is structured as follows: Section II presents the related work. Section III describes the system model and the assumptions we made in the formulation of our problem. Section IV presents the algorithm to build clustered robust configurations considering the time continuum, the traffic space and routing similarities. Numerical results are discussed in Section V. Finally, concluding remarks are presented in Section VI.

II. Related Work

The simplicity of controlling SDNs has brought back to light problems like mitigating and scheduling network reconfigurations [16], [17], since service providers are concerned about possible network outages caused by failures of route updates or sudden traffic changes. The networking research community has developed three classes of techniques to handle traffic change: (i) *dynamic TE*, which reconfigures the network each time a new event occurs, (ii) *static TE*, which uses a single precomputed configuration that minimizes the worst deviation to the optimum, and (iii) *semi-static TE*, which reconfigures the network at predefined time instants (e.g., twice per day at noon and midnight) to further improve network performance. Examples of *dynamic TE* includes methods like [3], [4], [6], [7], where sophisticated techniques are used to compute the best network configuration any time the traffic changes. However, reconfiguring the network too frequently can affect its stability, since programming hardware equipment with new flow rules can take longer than the reconfiguration period [4], thus causing the overflow of flow rules. Methods that reduce this burden have been proposed by prioritizing [18] or prefiltering [19] network updates. Nonetheless, the computation of each routing solution is not robust against prediction errors on the next TM.

One of the first techniques of *static TE* is oblivious routing [10], [20], and its recent extension called *valiant routing* [21], which randomly selects paths to connect source-destination pairs using a small subset of preselected intermediate nodes. Being totally oblivious to any traffic information, oblivious routing shows high performance loss as the network size grows. Exploring a partial knowledge of the traffic can reduce the performance loss. For example, COPE [13] considers only the most likely TMs for computing the optimal configuration and add a penalty term to avoid large deviation for less probable TMs. The technique proposed in [22] expands the most likely polytope by including TMs of normal operations in the direction of a predicted anomaly. The method proposed in [11] introduces different models for traffic uncertainness by expressing the maximum load that can be expected over a link in the pipe model or an upper bound on the traffic originating from a source node and directed to a destination node in the hose model.

Semi-static TE [14], [15], [23] provides a limited set of routing configurations with guaranteed performance loss. These works divide the TM polytope in two subsets according to the time dimension and compute a robust routing for each subset. While representing a first attempt to split the TM domain in multiple parts, these works present several limitations: (i) the slicing direction is arbitrary, (ii) the number of created subsets is limited, and (iii) the partition is performed either in the traffic domain or in the time domain.

Although semi-static TE approaches have the potential to optimize network performance using a limited set of routing configurations, traffic, time, and routing spaces/dimensions should be jointly considered when building clusters in order to avoid oscillations between routing configurations when TMs are close in the traffic space but far in the time dimension. Furthermore, clusters should not be sharply separated, since instantaneous routing changes are impossible even in SDNs. This work is a first attempt to address these problems and decide the best trade-off between reconfiguration rate and optimality of routing.

III. System Model

In this section, we present the traffic and routing system models that we consider in the design of our CRR algorithm.

We consider a system composed of two main stages: (i) a cluster-maintenance stage where we group TMs into clusters and compute robust routing configurations over these clusters, and (ii) a cluster-activation stage where we track the traffic evolution and reconfigure the network accordingly. The target is to minimize the Maximum Link Utilization (MLU) over time, which is motivated in the domain of datacenter interconnection and enterprise networks, where the goal is to minimize the network congestion.

We model the network infrastructure as an undirected graph $G = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ represents the set of network nodes and $\mathcal{L}$ models the set of links $e = (i, j)$, connecting network nodes $i, j \in \mathcal{N}$. Each link $e \in \mathcal{L}$ has a limited capacity c_{ij} that represents the maximum amount of traffic that the link can transmit. The set of active demands, also known as Origin-Destination (OD) flows, that need to be routed through the network, is represented as a Traffic Matrix (TM): a $|\mathcal{N}| \times |\mathcal{N}|$ matrix $T = [t_{ij}]$ where each element t_{ij} denotes the amount of traffic transmitted from source node i to destination node j. Since the traffic evolves over time, we consider a dynamic TM

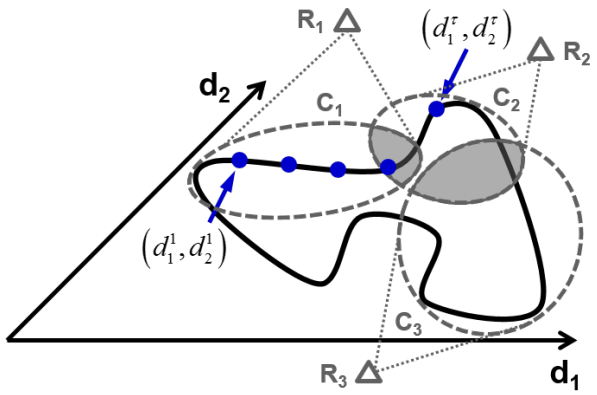


Fig. 1: Clustering of TMs. The solid black line represents the evolution of the two OD flows in the TM. Each blue point represents a sampled TM. Dashed ellipsoids denote the clusters of TMs, whereas triangles identify the corresponding routing configurations.

$T(\tau) = [t_{ij}^\tau]$, where τ denotes the time dimension. We assume that time is discretized and we have M samples of the TM (i.e., $\tau = 1, ..., M$). To simplify the notation, TM are usually represented as a $|\mathcal{N}|^2 \times 1$ demand vector $D(\tau) = [d_h^\tau]$ where each element d_h^τ unequivocally corresponds to an element t_{ij}^τ of the TM.

Cluster maintenance. Fig. 1 graphically illustrates the time evolution of a TM composed by only two demands, d_1 and d_2. The solid line represents the continuous evolution of the TM, whereas solid dots corresponds to periodic samples measured by a traffic monitoring system. As illustrated in the figure, an offline stage splits the TM domain into N clusters, denoted as C_i, and computes for each subset of TMs a routing configuration R_i, which is robust against any possible traffic variation within the cluster C_i. To avoid oscillations between routing configurations, a cluster C_i is built with a minimum time length L that results in a minimum utilization of the same routing configuration C_i. Furthermore, a temporal overlap O (the gray intersection in Fig. 1) is imposed between two adjacent clusters C_i and C_j to guarantee the feasibility of the corresponding robust routing configurations R_i and R_j outside their clusters. The overlap O provides further robustness against inaccurate cluster transition and can potentially leave some time to the real-time SDN controller decision on whether to reconfigure the network.

Cluster activation. The different precomputed routing configurations are then activated by the SDN controller which follows the evolution of the traffic matrix. By receiving an estimate of actual traffic conditions (and possibly a short term prediction) from the monitoring system, it can even decide whether to fetch and activate a better robust routing configuration in switches.

Clearly, the performance of this approach depends on the size and the number of clusters. Many small clusters result in a high reconfiguration frequency, which may harms the network behavior itself. In contrast, too few clusters will provide a low gain over static-TE solutions (e.g., oblivious routing). This approach will always lead to a better performance than the single-design case, because the network is no longer forced to always support the worst-case traffic demand. Indeed, the correct solution will be applied when the corresponding worst-case (among the possibly many that lie in different regions) appears.

Parameter	Description
$\mathcal{N}$	Nodes (network devices).
$\mathcal{L}$	Edges (network links).
c_{ij}	edge capacity (in capacity units) $(i,j) \in \mathcal{L}$.
d_h^τ	rate of OD demand d measured at time τ.
N	number of robust routing configurations.
M	number of traffic matrices.
L	minimum holding time of a routing configuration.
O	temporal overlap between two adjacent clusters.

TABLE I: Input parameters of our CRR algorithm.

Furthermore, a sharp boundary between clusters that are adjacent in the time domain requires an instantaneous reconfiguration of the network. For a smooth network reconfiguration when the TM enters into a new cluster, we compute clusters with a minimum time overlap, represented by the intersection of two clusters in Fig. 1.

In the next section, we show how our algorithm, Clustered Robust Routing (CRR), solves the problem of achieving a good trade-off between routing stability and optimality by maintaining a set of routing configurations for overlapping clusters of similar traffic matrices.

Table I summarizes the notation used throughout the paper.

IV. Clustered Robust Routing

The CRR algorithm is implemented as a module of the network controller. It takes as input a set of TMs representative of the period in which robust routing configurations should be designed. These TMs can be obtained in several ways: they can be measurements from past network conditions, or the outcome of a TM prediction module, or even synthetically generated. For the sake of clarity, we neglect the effect of prediction errors in the description of the algorithm, however, we investigate the impact of inaccurate TMs within numerical results. The impact is indeed rather limited for realistic error values because the clustering generates intrinsically robust solutions. Each TM describes the expected traffic conditions at specific time instants. Therefore, TMs can be temporally ordered and the set of TM IDs can be used as time axis. The result of the algorithm is a set of Routing Configurations (RCs, denoted as R_i in Fig. 1) and the corresponding clusters of TMs (C_i in Fig. 1). Each RC will be activated in the network as soon as the traffic enters the corresponding cluster.

A. Requirements for CRR

The CRR algorithm, shown in Fig. 3, consists in an iterative clustering and routing process relying on four main points:

a) Routing-based clustering: A TM clustering approach based on the similarity among the OD demands of each TM can be highly inefficient. Indeed, since network's links have limited capacity, good quality routes can substantially differ for TMs with similar demands. Since the way traffic

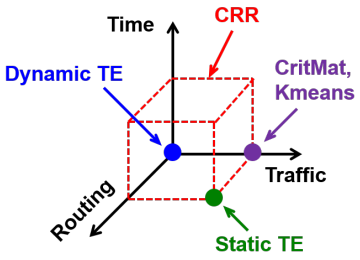


Fig. 2: Dimensions considered for the clustering of Traffic Matrices and computation of routing configurations. The traffic dimension is in reality multidimensional (one dimension for each OD flow).

is balanced over the network is not captured by the unique RC used to route the TM cluster, which is based on demand values, it may lead to high congestion for some TMs.

Things do not improve even if TMs with similar optimum routing are grouped together to generate a good unique cluster RC. Indeed, as we will show in Sec.V, this does not provide the best results. Due to scenario symmetries, different RCs can provide the same congestion, therefore clustering on the mere basis of RC topology can be largely suboptimal. Indeed, since the number of desired clusters in a solution is usually limited, this approach may waste clusters to separate TMs with different optimum routings, which could be equivalently well routed by another unique RC.

In order to better include the routing effects in the cluster selection, we need to consider the ultimate effect of the routing, that is the network congestion resulting from applying a given RC to a given TM. Only TMs that are characterized by a small congestion with the same RC must be grouped together into the cluster associated to the specific RC.

b) In-cluster robust routing: Although clustering is based on routing (i.e., it groups TMs having a similar congestion with a specific RC), the RC design cannot be strongly customized on a specific TM. Indeed, in practice we have to deal with deviation from the TM input set caused by imperfect measurements/predictions or even traffic anomalies. Therefore, we need a robust routing solution to cope with the demand uncertainty of the clustered TMs.

There are several approaches to robust routing in literature. The most straightforward solution is to consider the convex hull of all the discrete TMs and design the routing for the worst case in this continuous set. However, this approach has a number of drawbacks as the outcome may be strongly affected by a particular combination of demands that can be very rare in practice, thus producing an excessively conservative RC. In addition, the optimization process can be quite complicated [20], [24]. Since we assume a set of representative TM to be available, which represent the most likely or most important network conditions for the routing optimization process, we prefer to rely on a discrete space and adopt a multi-TM robust optimization approach, like those in [25]. A possible alternative when significant anomalies come into play is the approach presented in [13], in which the optimization process still focuses on the set of most representative TMs, while a bounded penalty gap is guaranteed over the remaining traffic domain, thus also in case of anomalies. However, a complete analysis of the anomaly management is out of the scope of this paper.

c) Routing configuration holding time: Although SDN provides flexible and efficient tools to dynamically change network routes, we must pay attention not to change the network configuration too rapidly, entailing route flapping problems. Therefore, the CRR algorithm includes for each activated RC a minimum holding time before reconfiguring. If the set of considered TMs is a uniform sampling of the expected traffic conditions, the minimum holding time is equivalently described by a minimum number of consecutive TMs in each cluster.

This feature brings in a new dimension in the clustering problem by adding the time dimension together with routing and traffic. Figure 2 illustrates graphically the design space of our algorithm and how the other approaches locate with respect to our proposal. Exploiting the time continuum, as well as the traffic and routing, it allows to improve the network performance and at the same time to reduce the number of network reconfigurations.

d) Adjacent clusters overlap: The transition of traffic conditions from those described by the current cluster to those of a new cluster must be carefully addressed to maintain a good routing quality. Although the technical route update process has been thoroughly studied and several SDN-based consistent update schemes are available [26], [27], a further fundamental issue is to decide when this update should occur. Different algorithms can be implemented to decide the best switching point depending on the past, current and predicted traffic behavior, considering anticipatory networking aspects as well. However, the common aspect among them will be an unavoidable uncertainty about the time to switch. Therefore, considering an overlap among adjacent clusters is important, because it guarantees a graceful transition between them. This means that RCs of adjacent clusters will be reasonably good with the TMs that are expected to be close to a route transition. This helps algorithms not to be too much penalized from suboptimal decisions.

Besides making reconfigurations robust to prediction errors, the time overlap facilitates the network reaction to traffic changes. SDN switches can store several RCs, the active one and a set of RCs potentially useful in the immediate future, according to traffic predictions. The time overlap allows to pre-fetch next RCs before reaching the cluster boundary, and thus, to anticipate reconfigurations (e.g., using TimeFlip [28]).

B. CRR design

In light of the above points, the problem we want to solve is to find the best assignment of M TMs to N robust RCs in order to find N TM clusters having a minimum length of L TMs and an overlap O. Moreover, since TMs' IDs are temporally ordered, the solution also provides the best expected cluster transition instants.

Note that the members of a cluster are required as input of the in-cluster robust routing, which, in turn is required to drive

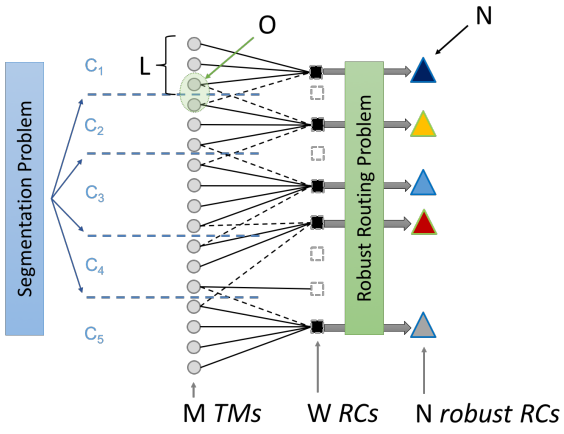


Fig. 3: High-level view of the proposed CRR algorithm.

the TM clusters formation, through the estimated congestion. Therefore, this two aspects must be jointly addressed to obtain an optimal solution. Unfortunately, the problem is strongly combinatorial and a joint optimization model has revealed to be very hard to solve. State-of-the-art integer programming solvers, like Gurobi Solver[1] or IBM CPLEX[2], could not provide a solution in reasonable times: small instances of tens of TMs require several days to get the optimum.

The hardness of the joint problem calls for the development of a heuristic approach to split the overall complexity in more affordable subproblems. In this perspective, we propose the two-step Clustered Robust Routing (CRR) algorithm represented in Fig. 3. In the first step, a *Segmentation Problem* is solved: the best assignment of M TMs to N out of W given RCs is computed, considering the minimum holding time constraint and the overlap. In the second step, a *Robust Routing Problem* is computed for each of the N clusters, in order to create new RCs better customized for the selected TMs. The new RCs are introduced in the set of available RCs to the Segmentation Problem and the two steps are repeated for a given number of iterations.

C. STEP 1 - Segmentation Problem

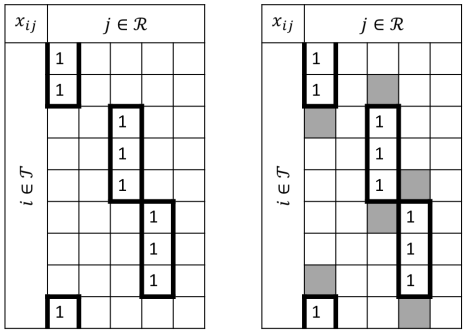
The Segmentation Problem takes in input a set of TMs $\mathcal{T} = T(1), ..., T(M)$ and a set of RCs $\mathcal{R} = R_1, ..., R_W$. Its goal is to assign each TM $T(i)$ to a RC R_j such that the overall association cost δ_{ij} is minimized and the number of used RCs is not larger than N. The cost δ_{ij} can be precomputed and corresponds to the network Maximum Link Utilization (MLU)[3] when TM $T(i)$ is routed through RC R_j. This creates a set of N TM clusters and RCs to manage the routing during the considered time period.

We model the Segmentation Problem as an ILP model based on two sets on binary variables. Variables $x_{ij}, i \in \mathcal{T}, j \in \mathcal{R}$,

[1]www.gurobi.com

[2]www.ibm.com/software/commerce/optimization/cplex-optimizer/

[3]Note that we decided to used MLU as it is a commonly used metric that directly expresses the network congestion, however the model is general enough to consider other types of metric



(a) Segmentation with no overlap

(b) Segmentation with $O = 1$

Fig. 4: Segmentation problem, x_{ij} matrix

setting the assignment of TM $T(i)$ to the cluster associated to RC R_j, and variables $z_j, j \in \mathcal{R}$, indicating with the value 1 that RC j is used to form a cluster. If we visualize the matrix corresponding to variables x_{ij} (see Fig. 4a), the solution of the problem is a set N row-sequences of 1's with minimum length L. These sequences must be unique when appearing in a column, and correspond to a set of TM clusters associated to RCs. In order to identify the beginning of each cluster, we rely on variables y_{ij}, $i \in \mathcal{T}, j \in \mathcal{R}$, forward differences of variables x_{ij}. When $y_{ij} = 1$, we can identify the initial TM of the cluster assigned to RC R_j: $T(i+1)$. The problem is fully described by the following ILP model:

$$[\mathbf{SP}] : \text{min.} \sum_{i \in \mathcal{T}, j \in \mathcal{R}} x_{ij}\delta_{ij} \qquad \text{s. t.:} \tag{1}$$

$$y_{ij} \geq x_{(i+1)_{|\mathcal{T}|}j} - x_{ij}, \qquad \forall i \in \mathcal{T}, j \in \mathcal{R} \tag{2}$$

$$\sum_{i \in \mathcal{T}} y_{ij} \leq z_j, \qquad \forall j \in \mathcal{R} \tag{3}$$

$$\sum_{i \in \mathcal{T}, j \in \mathcal{R}} y_{ij} \leq \sum_{j \in \mathcal{R}} z_j \tag{4}$$

$$\sum_{j \in \mathcal{R}} x_{ij} = 1, \qquad \forall i \in \mathcal{T} \tag{5}$$

$$\sum_{i \in \mathcal{T}} x_{ij} \geq L \cdot z_j, \qquad \forall j \in \mathcal{R} \tag{6}$$

$$\sum_{j \in \mathcal{R}} z_j \leq N \tag{7}$$

$$x_{ij}, y_{ij}, z_j \in \{0, 1\}, \qquad \forall i \in \mathcal{T}, j \in \mathcal{R} \tag{8}$$

The objective function (1) minimizes the sum of the association costs. The first set of constraints (2)[4] force variables y_{ij} to be 1 whenever the forward differences of variables x_{ij} are 1. Constraints (3) force the activation of the variable z_j, associated to RC R_j, if a non-null forward difference is present in column j, which means cluster C_j is considered in the solution. Constraint (4) states that the number of non-null forward differences in the matrix must not exceed the number of selected RCs. Therefore, together with constraints (3), this guarantees a unique cluster for each "active" column. The set

[4]The notation $(\cdot)_m$ indicates the modulo-m operator

of constraints (5) impose each TM $T(i)$ to be assigned to a unique RC, while constraints (6) force a minimum number of TMs associated to RC R_j, which, combined with the previous constraints imposing a unique and compact sequence of 1's within a column, correspond to constrain the minimum cluster length. Finally, constraint (7) states that no more than N clusters can be generated.

The set $\mathcal{R}$ is initialized by considering W RCs obtained by the solution of the Robust Routing problem over W sequential groups of TMs spanning the entire set $\mathcal{T}$. At the end of each algorithm iteration, the computed RCs are included in this initial set, this provides $\mathcal{R}$ with more refined RCs, which could be selected in the solution of the Segmentation Problem of the next iteration.

In order to consider an overlap between adjacent clusters, formulation **SP** must be amended to introduce the fact that up to O TMs beyond the boundaries of the clusters could be routed with the RC associated to the cluster. We model this by stating that each of the O TMs in overlap (grey cells in Fig. 4b) provides a congestion contribution δ_{ij} that is the average between the one of its associated cluster and the one of that in overlap[5]. The following constraints:

$$w_{ij} \geq x_{(i-1)_{|\mathcal{T}|}j} - x_{ij}, \qquad \forall i \in \mathcal{T}, j \in \mathcal{R} \tag{9}$$

$$\sum_{i \in \mathcal{T}} w_{ij} \leq z_j, \qquad \forall j \in \mathcal{R} \tag{10}$$

$$\sum_{i \in \mathcal{T}, j \in \mathcal{R}} w_{ij} \leq \sum_{j \in \mathcal{R}} z_j \tag{11}$$

and a new objective function must be introduced in **SP**:

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{T}, j \in \mathcal{R}} x_{ij}\delta_{i,j} + \\ & \frac{1}{2} \sum_{i \in \mathcal{T}, j \in \mathcal{R}} y_{ij} \left(\sum_{(i-O<k\leq i)_{|\mathcal{T}|}} \delta_{k,j} - \sum_{(i+1\leq k\leq i+O)_{|\mathcal{T}|}} \delta_{k,j} \right) + \\ & \frac{1}{2} \sum_{i \in \mathcal{T}, j \in \mathcal{R}} w_{ij} \left(\sum_{(i\leq k<i+O)_{|\mathcal{T}|}} \delta_{k,j} - \sum_{(i-O\leq k<i)_{|\mathcal{T}|}} \delta_{k,j} \right) \end{aligned} \tag{12}$$

Constraints (9)-(10) define variables w_{ij} as backward differences of x_{ij}. Interpreted as the end of the compact row-sequences of 1's in Fig. 4, w_{ij} must satisfy the same uniqueness requirements as y_{ij}. The new objective function (12) updates the association cost of TMs in overlap by removing half of the cost related to the associated cluster's RC and adding half of the cost towards the RC of the cluster in overlap.

D. STEP 2 - Robust Routing Problem

Once TMs have been clustered around an RC in STEP 1, STEP 2 computes a new robust RC R considering the TMs in the cluster. This will likely provide a better customized routing. In addition, being a robust routing, it makes CRR intrinsically robust against noisy TM measurements.

[5]The model can capture other assumptions by simply changing some of the coefficients in the formulation.

We compute R as robust RC that minimizes the MLU γ_{max} measured over network links, $(i,j) \in \mathcal{L}$, when the set of TMs in cluster C_c, denoted as $\mathcal{T}_c$, is routed via R. The TMs in $\mathcal{T}_c$ are characterized by the same demand set $\mathcal{H}$, but different demand values, varying according to the traffic time evolution. We express the unique RC via flow variables f_{ij}^h, which indicate the amount of demand h flow of every TM in $\mathcal{T}_c$ must be routed along the link (i,j).

The ILP formulation of the Robust Routing Problem is:

$$[\mathbf{RR}] : \text{min. } \gamma_{max} \qquad \text{s. t.:} \tag{13}$$

$$\sum_{(i,j)\in\mathcal{L}} f_{ij}^h - \sum_{(j,i)\in\mathcal{L}} f_{ji}^h = \begin{cases} 1 & \text{if } i = O_h \\ -1 & \text{if } i = D_h \\ 0 & \text{otherwise} \end{cases}$$

$$\forall i \in \mathcal{N}, h \in \mathcal{H} \tag{14}$$

$$\sum_{h \in \mathcal{H}} d_h^m \cdot f_{ij}^h \leq c_{ij}, \forall m \in \mathcal{T}_c, (i,j) \in \mathcal{L} \tag{15}$$

$$\gamma_{max} \geq \frac{\sum_{h\in\mathcal{H}} d_h^m f_{ij}^h}{c_{ij}}, \forall m \in \mathcal{T}_c, (i,j) \in \mathcal{L} \tag{16}$$

$$0 \leq f_{ij}^h \leq 1, \qquad \forall h \in \mathcal{H}, (i,j) \in \mathcal{L} \tag{17}$$

Constraints (14) are standard flow conservation constraints for splittable routing[6], with O_h and D_h, respectively, origin and destination of the OD demand h. Constraints (15) guarantee that the routing of each demand, with a request of d_h^m units of flow in TM $T(m)$, does not exceed the link capacity c_{ij} for any TM. Finally, constraints (16), together with the objective function, implement a min-max of the standard link utilization formulation at RHS of (16) over the TMs in $\mathcal{T}_c$.

V. Numerical Results

In order to assess the performance of the proposed algorithm, we consider a daily scenario in which we compare our CRR algorithm to different routing solutions within the Abilene Network [29], whose traffic requests are described by a set of TMs with granularity 5 minutes (288 TMs for the entire day). Abilene network was one of the first high-performance backbone networks, connecting 11 cities across United States. Nowadays, it is one of the very few real data sets in which network TMs and routing are public available. We imagine a scenario in which the optimization of clusters and RCs for the day after is run during the night, on the basis of daily TM predictions. Unless differently indicated, we average obtained results over a week and run the algorithm for 10 iterations. The CRR algorithm has been implemented in Python, using Gurobi Solver language interface.

A. Clustering approaches comparison

Fig. 5 shows the comparison of different clustering techniques, for the moment we do not consider the minimum cluster length constraint L and the overlap O. We measure

[6]We consider here a more general splittable routing because it can be easily implemented on SDN switches, however the model can be easily modified to consider unsplittable routing solutions as well.

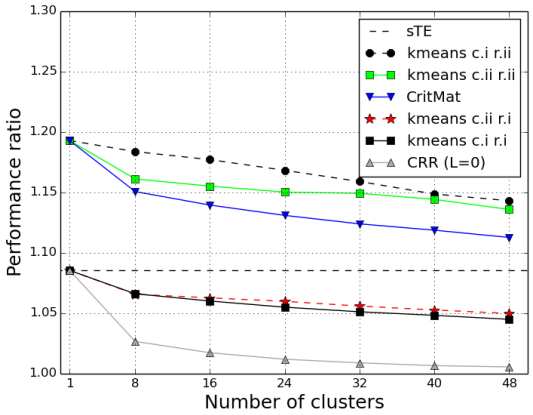


Fig. 5: Performance comparison of different TM clustering and robust routing approaches.

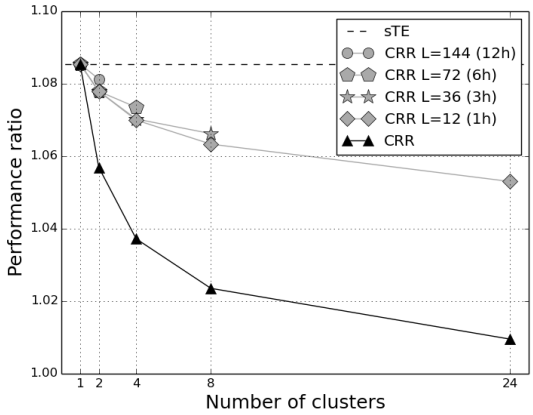


Fig. 6: Impact of different minimum cluster lengths on the CRR performance

the performance in terms of ratio of TM-averaged network MLU with respect to the value achievable by routing each TM through its MLU-optimum routing, that is, by applying dynamic TE. On the x-axis, the number of generated clusters is shown. We tested our CRR algorithm against different alternative approaches:

- *sTE*: A static TE solution where a robust routing is computed over the entire TM set. The result is a single daily RC and no reconfiguration is required, like in the case of oblivious routing.
- *CritMat*: This approach, presented in [15], consists in clustering TMs according to dominating cluster heads, which are synthetic TMs including the maximum of each demand among the TMs grouped into the cluster. The RC associated to the cluster is the MLU-optimum routing for the cluster head.
- *K-means clustering*: Most of the TM clustering approaches in literature are based on a variant of the well-known k-means technique. We have applied four k-means versions considering all combinations of the following clustering domains (*c*) and in-cluster robust routing approaches (*r*): *c.i)* clustering in the TM domain (considering similarity among OD demand values) and *c.ii)* clustering in the best-routing domain (considering similarity among MLU-optimum routing for each TM); *r.i)* robust routing applied as in formulation **RR**, *r.ii)* MLU-optimum routing applied to the dominating TM of each cluster.

We can note how the proposed CRR algorithm outperforms all other alternatives. The curves' trend shows that CritMat dominating TM appears to be over-conservative, as the resulting congestion is even worse than that of static TE. Indeed, the outcome RCs can address such a large set of potential TMs that their working points are largely suboptimal when RCs are applied to specific TMs. The k-means approaches show very different congestion levels. The type of applied in-cluster robust routing is the main performance driver: **RR** formulation provides remarkably better results than relying on a dominating TM. The impact of the clustering approach, instead, is limited and its benefit depends on the type of robust routing strategy subsequently applied.

Except for sTE, however, all the aforementioned clustering approaches have no constraints on the number of reconfigurations per time interval the network can be subject to. CRR has been applied with $L = 0$, CritMat can be shown to produce many RC changes. Even k-means approaches, although fixing the number of potential clusters, thus RCs, do not limit how many times they repeat and do not prevent reconfiguration bursts, where several RCs change in a short time interval. Therefore, we need to explicitly provide a minimum cluster length guarantee to avoid route flapping problems, which, as we will see, comes at the cost of a small congestion increase. This guarantee results in a fixed number of transitions, each separated by the desired length L.

B. Impact of minimum cluster length and overlap

In Fig. 6, we assess the performance of CRR algorithm when the minimum cluster length constraint is activated with different values of L. The x-axis shows the number N of clusters in a day, while different curves represent different values of L. Note that the values of L and N are not independent, as N clusters are generated in one day, N cannot be larger than the ratio 24 hours / L (in hours). Therefore curves with larger L stops at smaller N values. We can see that the minimum length constraint impacts on the performance of the clustering algorithm. With realistic N values, the MLU performance ratio increases from values about 1.02 (still referring to dynamic TE) to values about 1.06. CRR with $N = 8$ and $L = 36$ results in keeping the same routing configuration for at least 3 consecutive hours and changing only 8 times the RC during the next day.

Focusing on $N = 8$, the congestion of CRR with a minimum length $L = 1h$ is 6.3% higher than that in dynamic TE, which is better than the performance of the closest alternatives with no time constraints, 6.5%, that of the TM-domain k-means clustering with robust routing shown in Fig. 5. Therefore, our proposal, besides guaranteeing strong

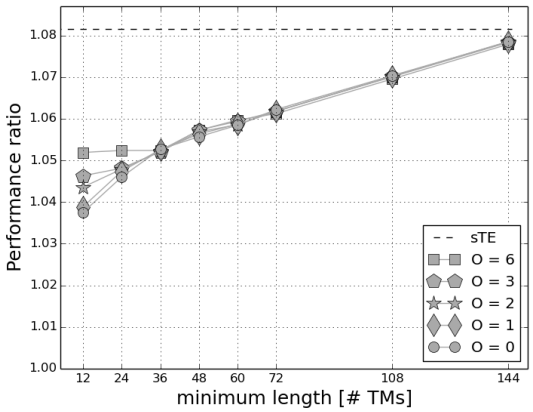


Fig. 7: Impact of different degrees of overlap on the CRR performance

bounds on the level of reconfiguration, allows to even decrease the network congestion. The quality of the clustering approach fully compensates the congestion increase caused by the minimum length constraint.

In Fig. 7, we analyze the performance of the CRR algorithm varying the degree of overlap O. The figure shows on the x-axis the minimum length L imposed to the cluster, while different curves are plotted for some values of O. We can see the impact of the overlap is significant only for short clusters, while it becomes quickly negligible when the minimum cluster length increases. Moreover, note that each TM included in the overlap provides an overlap extension of 5 minutes on each side. Therefore, considering $O = 1, .., 6$ means applying transition periods from 10 minutes to 1 hour, which we believe could reasonably include the complete set of meaningful values in practice, in terms of both uncertainness on the transition point and time required to anticipate reconfigurations.

C. Impact of prediction error

In the previous sections, we have analyzed the performance of the CRR algorithm when the clustering and the related RCs are computed over a set of TMs and applied to the same set. This corresponds to assume perfect TM prediction and provide the potential performance achievable by the algorithm. In this section, we relax this assumption and analyze the impact of prediction errors.

TABLE II: CRR performance when prediction error is considered. Results are expressed as percentage increase with respect to the ideal optimum routing.

cluster. \ α		0	15	30	45	60
sTE		6.52	7.02	8.19	9.25	**10.52**
CRR	L=72	4.07	5.13	6.93	**8.94**	11.09
	L=60	4.04	5.23	7.24	9.44	11.19
	L=48	3.76	5.06	7.12	9.61	11.79
	L=36	3.17	4.55	**6.74**	9.04	11.33
	L=24	2.84	4.50	6.85	9.35	11.93
	L=12	**2.06**	**4.29**	7.04	10.08	12.28

In order to reproduce the effect of unideal predictions, we run the CRR algorithm over a noisy version of the daily set of TMs to compute clusters and RCs, then apply the RCs to the original set of TMs, which represent the real traffic behavior. Each noisy TM version has been obtained from the original one by adding a uniform relative error $[-\alpha, \alpha]\,\%$ to every OD demand d_h^m. The results of these experiments are shown in Table II, where the performance achievable with different cluster lengths L and prediction errors α is reported. Similarly to previous analyses, the performance is computed as the percentage increase of the average network MLU with respect to the ideal case of applying dynamic TE in perfect prediction conditions.

We can clearly note that the performance of CRR is negatively impacted by the presence of prediction error, however the intrinsic robustness of the clustered approach limits the performance decrease. Even with large errors, the gap with respect to the ideal Dynamic TE is within 10-12%. The most interesting aspect to note is the parameters setting that provides the best performance, whose outcome is marked in bold in the table. The results show that the larger the error, the larger the clusters of the best solution. Indeed, when the quality of predicted TMs worsens, considering robust RCs computed over larger sets of TMs provides more robustness to any variation. A bigger variety of TMs included in the cluster used to generate a RC allows to better cope with traffic uncertainness. Taking this to extremes, when we have very low-quality predictions, no clustering can be helpful, because the representative set of TMs and the actual traffic will have little correlation. In these conditions, the Static TE approach, which builds a single RC considering all possible TMs in a day, is the best one can apply, as it generates the most robust RC. On the contrary, however, few and larger clusters lead to a bigger gap with respect to the dynamic TE, shown in Figg. 5 and 6, as the generated RCs are more conservative and far from being MLU-optimal for specific TMs. Therefore, a trade-off between cluster length and prediction accuracy exists. In case of good predictions, the size of the clusters drives the performance, vice versa, if predictions are affected by large errors, the impact of TM uncertainness completely overwhelms the effect of cluster sizes.

D. Open issues

This trade-off between cluster size and prediction accuracy opens a new technical challenge, which we cannot address in this paper, but appears to be a promising research direction. Thanks to the SDN paradigm, the controller can collect quasi-realtime measurements, predict the traffic evolution, and estimate a-posteriori the prediction error. It can act as an online mechanism able to anticipate the clusters that could be potentially visited in the near future and could provide the desired optimality gap with respect to an ideal congestion level. Moreover, the controller can prearrange a set of robust RCs derived from TM clusters, which, although referring to

the same TM centroid, are characterized by different sizes, i.e., robustness levels, so that the system can easily shift through different RCs when the prediction accuracy suddenly changes, as in case of anomalies. Accuracy changes can be detected by comparing link utilizations considered during the clusters' creation and those actually measured in the real network. Finally, clusters can be synthetically generated as well, in order to include potentially severe failures RCs must be robust against.

The SDN controller must play the main role in real-time managing the set of available RCs to orchestrate the routing of the entire SDN network over time by dispatching and activating the best RCs in each SDN switch. The design of the algorithm to select the type of generated RCs and the orchestration strategy is fundamental to provide performance optimality and full flexibility in front of traffic changes to advanced Software Defined Networks.

VI. Conclusion

In this paper we investigated how robust routing approaches can be made adaptive in the SDN context. Assuming the availability of traffic predictions, we designed an off-line method to split the traffic space into smaller partitions and build routing configurations that are robust against any real-time traffic variation within the partition.

The results showed that routing configurations based on TM clustering can achieve a performance very close to the optimal routing only if a good clustering domain is chosen. Our proposal based on the estimation of the congestion caused by the activation of a given routing outperforms the other candidate solutions. This good performance is also confirmed when the clustering is further constrained by technical and practical issues on the obtained routing configurations, which we considered in our approach.

Finally, we investigated the behavior of our solution when the accuracy of traffic predictions varies. It showed an interesting trade-off between cluster sizes and prediction errors that opens a new research direction for the orchestration of robust routing configurations over time in SDN.

References

[1] N. Wang, K. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Comm. Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM CCR*, vol. 43, no. 4, 2013, pp. 15–26.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 3–14, 2013.

[5] M. Malboubi, L. Wang, C. N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de)Aggregation and Measurement Paradigm (iSTAMP)," in *Proc. IEEE INFOCOM*, 2014.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. ACM CoNext*, 2011, p. 8.

[7] M. Roughan, M. Thorup, and Y. Zhang, "Traffic engineering with estimated traffic matrices," in *Proc. ACM IMC*, 2003.

[8] K. Murakami and H. S. Kim, "Optimal capacity and flow assignment for self-healing ATM networks based on line and end-to-end restoration," *IEEE/ACM Trans. on Networking*, vol. 6, no. 2, pp. 207–221, Apr 1998.

[9] C. Albrecht, "Global routing by new approximation algorithms for multicommodity flow," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, pp. 622–632, May 2001.

[10] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke, "Optimal oblivious routing in polynomial time," in *ACM Symp. on Theory of Computing*, 2003, pp. 383–388.

[11] V. Tabatabaee, A. Kashyap, B. Bhattacharjee, R. J. La, and M. A. Shayman, "Robust routing with unknown traffic matrices," in *Proc. IEEE INFOCOM*, 2007, pp. 2436–2440.

[12] M. Kodialam, T. Lakshman, and S. Sengupta, "Efficient and robust routing of highly variable traffic," in *Proc. HotNets*, 2004.

[13] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "COPE: traffic engineering in dynamic networks," in *ACM SIGCOMM CCR*, vol. 36, no. 4, 2006, pp. 99–110.

[14] P. Casas, L. Fillatre, and S. Vaton, "Multi Hour Robust Routing and Fast Load Change Detection for Traffic Engineering," in *Proc. IEEE ICC*, May 2008, pp. 5777–5782.

[15] Y. Zhang and Z. Ge, "Finding critical traffic matrices," in *Proc. IEEE DSN*, 2005.

[16] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever, "SWIFT: Predictive Fast Reroute," in *Proc. ACM SIGCOMM*, 2017.

[17] S. Brandt, K.-T. Förster, and R. Wattenhofer, "On Consistent Migration of Flows in SDNs," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[18] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM CCR*, vol. 44, no. 4, 2014, pp. 539–550.

[19] S. Paris, A. Destounis, L. Maggi, G. S. Paschos, and J. Leguay, "Controlling flow reconfigurations in SDN," in *Proc. IEEE INFOCOM*, 2016.

[20] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *Proc. ACM SIGCOMM*, 2003.

[21] R. Zhang-Shen, "Valiant Load-Balancing: Building Networks That Can Support All Traffic Matrices," *Algorithms for Next Generation Networks*, pp. 19–30, 2010.

[22] P. Casas, F. Larroca, and S. Vaton, "Robust routing mechanisms for intradomain traffic engineering in dynamic networks," in *Proc. IEEE LANOMS*, 2009, pp. 1–10.

[23] W. Ben-Ameur and M. Żotkiewicz, "Robust routing and optimal partitioning of a traffic demand polytope," *Intl. Trans. in Operational Research*, vol. 18, no. 3, pp. 307–333, 2011.

[24] W. Ben-Ameur and H. Kerivin, "Routing of uncertain traffic demands," *Optimization and Engineering*, vol. 6, no. 3, pp. 283–313, 2005.

[25] C. Zhang, Y. Liu, W. Gong, J. Kurose, R. Moll, and D. Towsley, "On optimal routing with multiple traffic matrices," in *Proc. IEEE INFOCOM*, vol. 1, 2005, pp. 607–618.

[26] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proc. Work. on Hot Topics in Networks*. ACM, 2011, p. 7.

[27] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[28] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Scheduling network updates with timestamp-based TCAM ranges," in *IEEE INFOCOM*, 2015.

[29] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural Analysis of Network Traffic Flows," *ACM SIGMETRICS PER*, vol. 32, no. 1, pp. 61–72, Jun. 2004.

An Online Power-Aware Routing in SDN with Congestion-Avoidance Traffic Reallocation

Adriana Fernández-Fernández*, Cristina Cervelló-Pastor*, Leonardo Ochoa-Aday*, Paola Grosso†
*Department of Network Engineering, Universitat Politècnica de Catalunya,
Esteve Terradas, 7, 08860, Castelldefels, Spain
Email: adriana.fernandez@entel.upc.edu, cristina@entel.upc.edu, leonardo.ochoa@entel.upc.edu
†System and Network Engineering group (SNE), University of Amsterdam,
Amsterdam, The Netherlands
Email: p.grosso@uva.nl

***Abstract*—Software-Defined Networks (SDN) can be seen as a promising alternative to achieve the long-awaited power efficiency in current communications systems. In these programmable networks a power-aware mechanism could be easily implemented leveraging the capabilities provided by control and data plane separation. For such purpose, this paper proposes a novel solution minimizing the number of active elements required in an SDN with multiple controllers and in-band control traffic. In order to provide a complete and fine-grained strategy, this proposal comprises two crucial modules: GrIS, a green initial setup and DyPAR, a dynamic power-aware routing. Besides being compatible with SDN environments without a dedicated control network, the proposed strategy is able to handle demanding traffic arrival without degrading the performance of higher priority traffic. Simulation results show that our heuristic approach allows to obtain close-to-optimal power savings with differences under 8%. Moreover, comparison with existing related methods using real topologies validates the improvements achieved by our solution in terms of power efficiency and performance degradation avoidance. For instance, after routing all the incoming traffic, a reduction of power consumption of up to 26.5% and an increase of allocated demands of up to 26.7% can be reached by our solution.**

I. Introduction

Energy consumption concern in communication systems has currently attracted a great deal of attention from research community due to the exponential demand growth and the ever-increasing number of connected devices [1]. According to [2] by 2025 the global Internet will be responsible for more than 10% of the world's electricity consumption. Given that in practice power consumption of network equipment is not in proportion with their traffic load, putting unused network elements into sleep mode (i.e. a low-power state) is an effective and widely accepted strategy to decrease the consumption of data networks [3].

In this context, Software-Defined Networking (SDN) is a very well-suited architecture to perform power-aware routing and manage the state of unused switch interfaces in a coordinated and centralized way. The basic idea of SDN [4] -control and data planes separation- makes network environments more manageable. The logically centralized control plane in SDN provides a global knowledge of the network state information. Moreover, it is responsible for managing network tasks and perform device programming. Meanwhile, interconnection devices only follow the rules set by the controller to forward the traffic. Therefore, the implementation of a power-aware solution in the control plane is a valuable opportunity to solve the power consumption problem in data networks, making it easier than with classical hardware-dependent standards.

Despite consistent efforts to improve the network power efficiency, power-aware techniques may lead to performance degradations. Inspired by this reality, this paper introduces a new power-aware strategy combining a control plane configuration with a dynamic routing for an SDN architecture. This solution dynamically reduces the number of active nodes and links required to manage changing traffic patterns. Instead of restricting the potential of power-aware solutions to low-loaded environments, this work proposes a more fine-grained strategy minimizing the power consumption while avoiding the performance degradation of higher priority traffic.

Throughout this work we consider an SDN architecture with multiple controllers and in-band control traffic [5]. In this operational mode, links are shared between data and control plane traffic. Hence, the proposed power-aware routing can be applied also in cases when implementing a dedicated control network is not feasible either for physical or cost-related restrictions. In backbone networks this is a more realistic scenario since additional links dedicated to directly connect controllers and forwarding devices, are impractical and cost-inefficient. Specifically, the major contributions of this work are as follows:

- An Integer Linear Problem (ILP) is formulated to optimize the number of active nodes and links in SDN with multiple controllers and in-band control traffic.
- A novel power-aware mechanism is proposed to allocate traffic demands in real time, reducing power consumption and performance degradation of higher priority traffic.
- Real topologies, as well as existing related proposals, are used to validate achieved improvements.

This work was supported by the Ministerio de Economía y Competitividad of the Spanish Government under project TEC2016-76795-C6-1-R and AEI/FEDER, UE and through a predoctoral FPI scholarship.
ISBN 978-3-903176-08-9

II. Related Works

Throughout recent years the power consumption of communication networks has been extensively treated and several solutions focused on reducing the number of active elements have been proposed. For instance, Bianzino et al. [6] aim to find the network configuration that minimizes the network energy consumption, modeled as the sum of the energy spent by all nodes and links carrying traffic. To achieve this, they formulated an optimization problem for finding minimum-power network subsets assuming the existence of traffic level with known daily behavior. Therefore, an accurate prediction of incoming traffic is required.

An energy-aware routing and traffic management solution is proposed in [7] to reduce the energy consumption, determined as the number of active Open-Flow switches in the network. For this, a low complexity algorithm is presented using, for each pair of endpoints, a pre-computed set of shortest paths to select the route that minimizes the number of switches that become active after allocating the flow. Although this proposal allows real-time operation routing flows sequentially, only low-loaded nighttime traffic is considered, failing to extensively examine the implications of more demanding scenarios.

The authors of [8] presented the design of an Energy Monitoring and Management Application (EMMA) to minimize energy consumption in SDN-based backhaul networks. They formulated this problem as a non-linear optimization model and proposed heuristic algorithms for the dynamic routing of flows and the management of the resulting link and switch activity. However, such algorithms were implemented in an SDN emulation environment with out-of-band control traffic, limiting their applicability to networks where dedicated links between the controller and forwarding devices are deployed.

In [9] authors proposed ElasticTree, a network-wide power manager to save energy in data centers using SDN. This solution dynamically finds the minimum set of network elements required by changing traffic loads, while satisfying performance and fault tolerance constraints. In this regard, three strategies were studied, namely Formal Model, Greedy Bin-Packing and Topology-aware Heuristic. While the first option presents scalability issues and the second saves less power, the best performance is obtained by the Topology-aware Heuristic. However, this approach is specifically conceived for FatTree networks.

Other approaches about power efficiency in software defined data center networks are presented in [10], [11]. The authors of [10] simultaneously optimize the power saving and the network performance, according to a pre-defined combination of quality requirements. In [11] different energy-aware routing strategies, combining common routing and scheduling algorithms, are evaluated and implemented as a OpenNaaS-based prototype. However, these strategies are only applicable in data centers and are also incompatible with environments without dedicated control networks.

Different from the aforementioned works, the aim of this paper is to provide a power-aware control plane configuration combined with a dynamic routing strategy considering an SDN architecture with multiple controllers and in-band control traffic. Our approach is also able to handle more demanding traffic patterns while reducing the performance degradation of higher priority traffic.

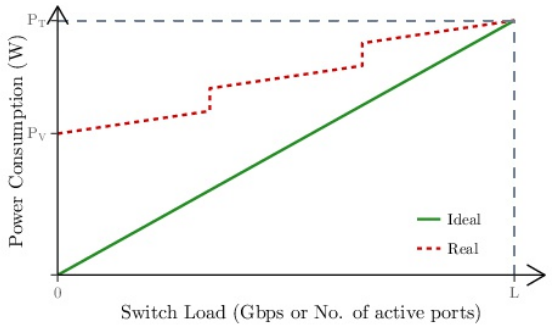


Fig. 1. Power Model (redrawn from [12]).

III. Power Model

Power consumption of networking devices is composed by a static component (due to power consumed by chassis, fans, line-cards, etc.) and a dynamic one, related to the rate of traffic flowing through their port interfaces. Ideally, the static part, also known as the idle component, which represents the power required by an unused switch, should be null. Then, in presence of an increasing traffic load, the power consumption should behave proportionally and linearly grow along with the traffic increase (line marked as Ideal in Fig. 1). However, this model differs considerably from the real one (line marked as Real in Fig. 1). In practice, whenever a device is active it will consume a fixed amount of power (P_n), irrespective of load conditions. Additionally, this baseline power is increased by the number of active ports and the utilization of each port.

In this regard, it has been previously measured that the amount of traffic handled by port interfaces does not have a significant effect on device's consumption [12]. Explicitly, while most of the power is consumed only by turning the device on, increasing the port utilization from zero to full load represents less than 8% of total power consumption [9]. Therefore, in this paper we consider that the power consumed by a network node depends on the baseline power and the number of active ports, both of which represent fixed contribution.

IV. Problem Statement

To formalize the power consumption optimization problem in SDN, in this section we present its mathematical formulation. The proposed model seeks to optimize the overall power consumption. To that end, the incoming traffic demands and the associated required control traffic will be routed minimizing the number of active network elements. In general, our model leverages preliminary works presented in [13]–[15] supporting that forwarding devices are put into sleep mode.

Being a general formulation, multiple controllers as well as SDN with in-band mode are supported by this proposal.

Given the controllers placement, our model also determines the optimal distribution of switches between controllers in terms of power efficiency and load balancing.

A. Network Model

In the proposed scheme the network topology can be modeled as a graph $G = (V, E, C)$, where V, E and C denote the set of switches, links and controllers respectively. Note that network nodes can only fulfill one role, i.e. controller or routing device. Additionally, we use $c_{i,j}$ to denote the capacity of a link $(i, j) \in E$. Considering F as the entire set of traffic flowing through the network between any pair of nodes, let D denote the subset corresponding to data plane communications. For the control plane, we use T to denote the subset of communications between controllers and switches, and H to denote the subset of communications between controllers. Each flow $f \in F$ from source s_f to destination t_f, has associated a throughput, denoted by d_f.

B. Formulation

Considering the entire set of demands fixed and known in advance, all the optimal control and data paths in terms of power efficiency can be computed jointly in a global optimization process. To formulate such optimization problem, the required variables, objective functions and constraints are defined as follows:

TABLE I
NOTATION OF BINARY VARIABLES

Name	Description
$x_{i,j}$	Indicates whether link i, j is active
y_v	Indicates whether node v is active
$t^f_{i,j}$	Indicates whether link i, j is selected to route flow f
$\lambda_{v,c}$	Indicates whether node v is associated with controller c

The objective function of our model seeks to reduce the overall power consumption considering the number of active nodes and links in the network. Consequently, both elements are integrated in the following expression, where P_p and P_n denote the power consumption of a port and a node, respectively.

$$\text{minimize} \quad 2P_p \sum_{(i,j)\in E} x_{i,j} + P_n \sum_{v\in V} y_v \tag{1}$$

A single controller must be selected to manage each active forwarding device in the network.

$$\sum_{c\in C} \lambda_{v,c} = y_v \qquad \forall v \in V \tag{2}$$

Looking to avoid congested controllers, we set the maximum number of forwarding devices that can be associated with each controller. In this way, active switches are evenly distributed and the load is balanced among controllers.

$$\sum_{v\in V} \lambda_{v,c} \leq \left\lceil \frac{\sum_{v\in V} y_v}{|C|} \right\rceil \qquad \forall c \in C \tag{3}$$

A node $v \in V$ is active if there is traffic in any of its incoming or outgoing edges, being $N(v)$ the set of neighbors of v.

$$y_v \geq \frac{1}{2\,|F|} \sum_{f\in F} \left(\sum_{u\in N(v)} t^f_{u,v} + \sum_{u\in N(v)} t^f_{v,u} \right) \quad \forall v \in V \tag{4}$$

To avoid additional traffic load through network controllers, data plane communications (i.e. $f \in D$) cannot be routed through these devices. Furthermore, control traffic between controllers and switches (i.e. $f \in T$) will not pass through any other controller that is not the source or target of the traffic. The same must hold true for communications between controllers (i.e. $f \in H$). In these constraints we use $N(c)$ to denote the set of neighbors of a controller $c \in C$ and v_f to identify the forwarding device involved in the source/target pair of traffic flow $f \in T$.

$$\sum_{n\in N(c)} t^f_{n,c} \leq \begin{cases} 0 & \forall f \in D, \forall c \in C \\ \lambda_{v_f,c} & \forall f \in T, \forall c \in C \\ 0 & \forall f \in H, \forall c \in C \setminus \{s_f, t_f\} \end{cases} \tag{5}$$

The routing of data plane communications and control traffic exchange between controllers, follows the traditional flow conservation constraints.

$$\forall v \in V, \forall f \in D \cup H: \tag{6}$$
$$\sum_{u\in N(v)} t^f_{v,u} - \sum_{u\in N(v)} t^f_{u,v} = \begin{cases} 1 & \text{if } v = s_f \\ -1 & \text{if } v = t_f \\ 0 & \text{otherwise} \end{cases}$$

Meanwhile, for the subset of communications between controllers and switches $f \in T$, these constraints are modified to assure that only active switches exchange control messages with its controller. Similarly, the forwarding device and the controller involved in the source/target pair of traffic flow $f \in T$, are denoted with v_f and c_f, respectively.

$$\forall v \in V, \forall f \in T: \tag{7}$$
$$\sum_{u\in N(v)} t^f_{v,u} - \sum_{u\in N(v)} t^f_{u,v} = \begin{cases} y_v \lambda_{v_f,c_f} & \text{if } v = s_f \\ -y_v \lambda_{v_f,c_f} & \text{if } v = t_f \\ 0 & \text{otherwise} \end{cases}$$

Finally, a link (i, j) is active if it is used by some traffic flow $f \in F$. Furthermore, the total traffic in each active link must be less than its assigned capacity.

$$\sum_{f\in F} t^f_{i,j} d_f \leq c_{i,j} x_{i,j} \qquad \forall (i,j) \in E \tag{8}$$

Although this model allows the attainment of optimal solutions for the power consumption problem in SDN, it becomes challenging to solve on large and even medium-scale topologies. This is because the difficulty of the power-aware routing problem is known to be NP-Hard [16], so the consumption of resources and time complexity grow exponentially with the network size. To overcome this issue, in the next section we develop a heuristic algorithm.

V. Heuristic Algorithms

To compute all the routes (i.e. for data and associated control traffic) using the global optimization model presented previously, the entire set of traffic demands need to be fixed and known in advance. Considering this as a limitation for current dynamic networking environments, in this section we propose a new approach to support time-variable traffic requirements. The key idea of this proposal is to fully take advantage of the high control flexibility given by the dynamic configuration capabilities and centralized network view of SDN without needing an accurate prediction of incoming traffic. In order to allow that nodes are put into sleep mode we assume network topologies with forwarding devices divided into two categories: edge nodes, which are connected to some traffic source/target and transit nodes, which merely route other nodes traffic.

A. Green Initial Setup (GrIS)

An initial control plane configuration, previous to the data traffic arrival, is required in order to support the in-band mode in SDN. This control plane setup is intended to establish the communication paths between switches and controllers, as well as between controllers. In this way, when new traffic flows arrive, switches can send to the controller routing requests through packet_in messages. To do so, in this section we propose an off-line solution denoted as Green Initial Setup (GrIS). This component will be statically activated at specific time instances as a planned operation.

The proposed strategy, shown in Algorithm 1, takes as inputs the original network topology G with controller placements, the subset of edge nodes $S \subseteq V$ and the control traffic requirements R^c. The outputs are a reduced graph with the initially active network elements $G^A = (V^A, E^A, C)$, an array keeping the controller-switch associations A and the initially required control paths P_c.

In the first step, the algorithm reduces the number of initially active nodes using the NET_PRUNING function, shown in Algorithm 2. This method aims to remove as many nodes as possible, considering as candidates the set of network nodes that will not be endpoints of incoming demands, denoted in the pseudocode as N. For each node inside this set of transit nodes, the function computes its betweenness centrality (B_n), as a measure of its intermediary role in the network. In the proposed approach, we use a simplified version of this metric considering only the shortest paths from each controller to every switch. In particular, after computing the shortest paths from each controller as single source, the B_n associated with a node n is increased for each path containing the node (lines 5-14). Using these values, transit nodes are sorted and stored in the list N'. At each iteration of this list the function attempts to increase the number of switched-off nodes. A new node is removed only when in the resulting graph forwarding devices remain being reachable by network controllers, i.e. at least one path exists between every controller-switch pair in the network.

Algorithm 1 GrIS Pseudocode

Require: G, S, R^c
1: $G' \leftarrow$ NET_PRUNING(G, S)
2: $O \leftarrow$ Get_All_Admissible_Control_Paths(G', R^c)
3: $S' \leftarrow S$ sorted by nodes priority criteria
4: $s \leftarrow$ First node in S'
5: $\left|V^A\right|, \left|E^A\right| \leftarrow \infty$
6: **repeat**
7: **for** $p \in O[s]$ **do**
8: Initialize $(V^{A'}, E^{A'}, P'_c, A', U')$
9: **for** $u \in p \setminus S$ **do**
10: Power_Aware_Path_Selection$(O[u])$
11: Update $(V^{A'}, E^{A'}, P'_c, A', U')$
12: **end for**
13: **for** $n \in L \setminus f$ **do**
14: $r =$ Power_Aware_Path_Selection$(O[n])$
15: Update $(V^{A'}, E^{A'}, P'_c, A', U')$
16: **for** $v \in r \setminus S$ **do**
17: Power_Aware_Path_Selection$(O[v])$
18: Update $(V^{A'}, E^{A'}, P'_c, A', U')$
19: **end for**
20: **end for**
21: **for** $(c1, c2) \in G'$ **do**
22: Power_Aware_Path_Selection$(O[c1, c2])$
23: Update $(V^{A'}, E^{A'}, P'_c, A', U')$
24: **end for**
25: **if** $\left|V^{A'}\right| \leq \left|V^A\right| \wedge \left|E^{A'}\right| \leq \left|E^A\right|$ **then**
26: $V^A, E^A, P_c, A, U \leftarrow V^{A'}, E^{A'}, P'_c, A', U'$
27: **end if**
28: **end for**
29: **if** $\left|V^A\right| = \infty \vee \left|E^A\right| = \infty$ **then**
30: **if** $s =$ last node in L **then** break
31: **end if**
32: $s \leftarrow$ Next node in S'
33: **end if**
34: **until** $\left|V^A\right| \neq \infty \wedge \left|E^A\right| \neq \infty$

To accomplish this, a temporal graph, denoted as G', is created. This graph is used to check the required connectivity between controllers and forwarding devices. After validating that the possibility of reaching every node in the network from any controller is not affected, the considered node is removed from the resulting graph. This means that these nodes together with their links are put into sleep mode in the original graph.

After pruning the network, the GrIS algorithm uses the reduced graph G' to find the overall set of admissible control paths which satisfy control traffic requirements R^c (line 2 in Algorithm 1). As previously stated, these paths do not pass through any other controller that is not the source or target of the traffic. Using these computed control paths, a sorted list of ingress and egress forwarding devices is stored in S'. This list is sorted in ascending order following two priority criteria:

1) the number of possible controllers to associate with,
2) the number of possible control paths.

Algorithm 2 NET_PRUNING

Require: G, S
1: $G' \leftarrow G$
2: $N \leftarrow V - S$ ▷ Transit nodes
3: $H \leftarrow$ NULL ▷ Removed nodes
4: $B \leftarrow$ NULL ▷ Array of betweenness values
5: **for** $n \in N$ **do**
6: $B_n = 0$
7: **for** $c \in C$ **do**
8: $SP_c \leftarrow$ Set of shortest paths from controller $c \in C$
9: **for** $p \in SP_c$ **do**
10: **if** $n \in p$ **then** $B_n = B_n + 1$
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: $N' \leftarrow N$ sorted according to increasing order of B
16: **for** $n \in N'$ **do**
17: Remove from G' node n
18: **if** nodes are still reachable by controllers in G' **then**
19: Save n in H
20: **else**
21: $G' \leftarrow G$
22: Remove from G' nodes in H
23: **end if**
24: **end for**

Going through this list, the algorithm starts satisfying the most critical cases and the solution can be found with fewer iterations. The main loop of the Algorithm 1 determines for each possible control path of the selected node s, the number of active elements in the network after computing all control routes. The configuration of paths with fewer active elements is then selected in this process.

Inside this loop the algorithm first determines the paths between controllers and forwarding devices. Note that, for each forwarding device x, $O[x]$ contains admissible control paths to each controller available in the network. Precisely, paths selected in this step define one controller for each forwarding device. Additionally, any time a path between a switch and a controller is computed, nodes belonging to the control path but not in S are analyzed by the algorithm. Note that these nodes are the transit nodes that remained in the resulting graph after pruning the network. Since they are used to route some traffic, a control path is also established between them and some controller. After determining all switch-controller associations, the algorithm searches the paths between controllers.

In general, the power-aware path selected for every control pair is the best route between them in terms of minimizing the number of active elements in the network as long as it has sufficient link capacity to route the traffic volume, under the considered Maximum Link Utilization (MLU) constraint. Additionally, during the selection of one control path between each forwarding device and one controller, the number of devices already attached to the controllers is considered in order to keep a balanced load. Since the set of admissible paths obtained from the pruned network with a reduced number of elements is significantly smaller than in the original topology, the solution can be found with fewer iterations.

If after analyzing all control paths of node s, the algorithm cannot find a feasible configuration of paths to route all control and data plane communications, the main loop repeats this process for the next node stored in S'. This is done until the solution is found or until all forwarding nodes are analyzed, i.e. when the algorithm breaks without a solution. Note that this last option occurs when, given a controllers placement, an admissible configuration for controller-switches association could not be found or when the network has not sufficient capacity to meet the demand requirements under established constraints.

B. Dynamic Power-Aware Routing (DyPAR)

When a new traffic demand arrives, a routing request is sent from the input node to its associated controller using the previously computed path between both devices. Based on its global knowledge of the network topology, this controller calculates the required data path minimizing the number of elements that need to be activated for this connection request and creates the flow forwarding rules. The proposed dynamic power-aware routing, denoted as DyPAR and shown in Algorithm 3, performs in essence three crucial functions:

1) Power-aware data and control path selection;
2) Performance-aware data path selection;
3) Congestion-aware traffic reallocation.

For each incoming demand d, the algorithm starts trying to get the set of admissible data paths across the current active topology. This is done considering that admissible data paths do not pass through any controller in the network. If several paths were found, the one with the highest remaining available link capacity is selected. In this way, the number of future requests that can potentially be accommodated over the currently active paths is increased. Then, traffic is allocated and links utilization are updated.

On the other hand, if no admissible data path was found to route the incoming traffic across the currently active topology, the original network graph is then considered by the algorithm. Since the now determined candidate routes will imply the use of additional network elements, the data path minimizing the number of active network elements is selected to carry the demand. After updating the active topology and links utilization, a loop is used to establish the required control plane communications for each added node along the data path. In the same way, the algorithm first considers the currently active topology to set the required control path with some network controller and the entire network in case of failing the initial attempt.

In case of incoming traffic rates exceeding the remaining available network capacity (line 19 to 21), the algorithm considers all data paths in the original network without taking into account the capacity restrictions, but keeping that data plane communications cannot be routed through network controllers.

Algorithm 3 DyPAR Pseudocode

Require: G, G^A, P_c, A, U, d
$P_d \leftarrow$ Get_Admissible_Paths(G^A, d)
if $P_d \neq Null$ **then**
  $p_d \leftarrow$ Lest loaded path in P_d
  Update U after routing p_d
else
  $P_d \leftarrow$ Get_Admissible_Paths(G, d)
  if $P_d \neq Null$ **then**
    $p_d \leftarrow$ Power_Aware_Path_Selection(P_d)
    Update G^A, U after routing p_d
    for node n added to G^A by p_d **do**
      $P_c \leftarrow$ Get_Admissible_Paths(G^A, n, C)
      if $P_c = Null$ **then**
        $P_c \leftarrow$ Get_Admissible_Paths(G, n, C)
      end if
      $p_c \leftarrow$ Power_Aware_Path_Selection(P_c)
      Update G^A, U, A after routing p_c
    end for
  else
    $P_d \leftarrow$ Get_All_Paths(G, d)
    $p_d \leftarrow$ Performance_Aware_Path_Selection(P_d)
    Update U, T after routing p_d
  end if
  $BL \leftarrow$ Link with maximum load
  $F \leftarrow$ Demands established through BL
  CONGESTION_AWARE_REROUTING(G^A, F, BL, U)
end if

Then, the algorithm performs a data path selection based on reducing the performance degradation incurred. More in detail, the algorithm selects the data path inside this group whose congested links are less used by previously established demands. The reason is that, to allow the new traffic flow, the capacity remaining on those links, after allocating the QoS sensitive demands and control traffic, will be fairly divided between the involved lower-priority demands. Rates beyond this resulting throughput will be reduced and traffic will be handled on a "best-effort" basis. In this way, the proposed algorithm can efficiently handle bursty traffic and accommodate rates that exceed the remaining available capacity without affecting the QoS sensitive traffic if the network is not heavily loaded.

Every time a new network element is added to the active topology, the algorithm tries to alleviate the congestion level on the network. To accomplish this, after identifying the bottleneck link and the group of traffic flows going through this link, a CONGESTION_AWARE_REROUTING is performed. This function, described in Algorithm 4, starts creating a temporal graph G'' where the most loaded link is removed. Additionally, currently established demands sharing the most loaded link are sorted in decreasing order of rate requirements with the aim of reducing the congestion after rerouting the fewer number of connections. In order to avoid frequent reallocations of a traffic flow and mitigate related negative implications, a time threshold can be easily included to select

Algorithm 4 CONGESTION_AWARE_REROUTING

Require: G^A, F, BL, U
$Current\_MaxU \leftarrow U[BL]$
$G'' \leftarrow G^A$
Remove BL from G''
$F' \leftarrow F$ sorted by decreasing order of flow rate
for established demand f in F' **do**
  $P \leftarrow$ Get_Admissible_Paths(G'', f)
  $p \leftarrow$ Congestion_Avoidance_Path_Selection(P)
  $MaxU_p \leftarrow$ Maximum link utilization in p
  if $p \neq None \wedge MaxU_p < Current\_MaxU$ **then**
    Reroute f and associated control traffic
    Update U and $Current\_MaxU$
  end if
end for

only demands that have been allocated long enough over the current path. Using the residual graph a new set of admissible paths is obtained for each involved traffic flow. Then, the function looks for a path with lower load values trying to leave more resources available for future demands. A traffic flow is reallocated only when a feasible path is found and the MLU in the network is reduced. At the same time, the required control paths are updated.

Since the proposed approach is conceived for dynamic traffic environments, the set of established demands will be constantly checked. For those connection requests whose holding times have expired, the algorithm performs a demand removal, which means that their assigned paths are released and resources occupied by these routes become available again. Consequently, network elements used only by completed traffic demands will be then put into sleep mode.

VI. PERFORMANCE EVALUATION

To assess the performance of the ILP model, we used the linear programming solver Gurobi Optimizer [17]. Meanwhile, heuristic algorithms were implemented using the programming language Python. All computations were carried out on a computer with 3.30 GHz Intel Core i7 and 16 GB RAM.

A. Simulation Scenario

1) Network Topology: We conducted our simulations using real-world network topologies collected from SNDlib [18], considering each router in the network as an SDN node or as a controller placement. Specifically, in order to assess the effectiveness of the proposed scheme we use three topologies of different sizes. These networks are: Nobel-US (14 nodes, 21 links), Geant (22 nodes, 36 links) and Cost266 (37 nodes, 57 links). To allow the possibility of putting network nodes into sleep mode, different scenarios were considered varying T, which represents the percentage of forwarding devices that will not generate or receive traffic. According to this value, for each network topology we have selected as transit nodes the devices with the highest degree centrality as in [6].

2) Controllers Placement: Being the controller placements out of the scope of this paper we assume as preferred locations the ones minimizing the worst-case mean latencies. More precisely, we compute the mean propagation latency between each pair of nodes and associate each admissible location with the maximum average value involving it. Then, according to the number of controllers considered for each simulation instance, we place the controllers at node locations with smaller associated latency values. Note that a controller placement is admissible when the assumptions established in this proposal to avoid the routing of additional traffic load through network controllers can be kept (i.e. the network graph without any controller remains being strongly connected).

3) Traffic Patterns: Apart of the real static traffic matrices obtained from the topologies database in [18], we also consider a dynamic scenario where connection requests arrive with exponentially distributed inter-arrival and holding times taking different mean values from the sets [0.2, 1, 5] and [100, 150, 200], respectively. Accordingly, a traffic flow is generated between each pair of edge nodes (i.e. network devices which do not act as controllers or transit nodes). To evaluate the power savings and performance degradations considering increasing loads, for each network topology we considered every pair of edge nodes with an initial randomly assigned data rate and computed the associated shortest paths. We then identified the most loaded link from which we derived a scaling factor. Lastly, the initially assigned values were multiplied by this scaling factor to obtain the corresponding data rates for each incoming demand (see [19]). This was done considering different values of the over-provisioning factor (α) to further evaluate the implications of varying traffic load. We assume an average control traffic rate of 1.7 Mbps [20].

4) Power Values: Based on the power consumption behavior of data networks explained in Section II, we characterize the power consumption of a forwarding device using the 3:1 idle:active ratio given in [9]. This proportion, obtained from measurements on real switches, assigns 3W of power for each idle port of a switch and 1W extra when the port is active. Thus, power consumption P_n of a idle forwarding device n can be computed as $3D(n)$ where $D(n)$ denotes the node degree and P_p = 1W. Null power consumption is assumed when the node is put into sleep mode.

B. Optimal vs. Heuristic Solutions

To assess the suitability of the proposed solution we start evaluating the performance of the heuristic algorithms against the optimal ILP model, using the Nobel-US and Geant topologies with traffic matrices provided in [18]. This comparison is illustrated in Fig. 2 for different amount of controllers and percentage of transit nodes. Power savings are computed according to the expression $(Overall\_Pw - Pw\_X)/Overall\_Pw$, where $Overall\_Pw = \sum_{n \in V} P_n + 2P_p\,|E|$ and Pw_X is the power consumption achieved by the considered approach.

In Fig. 2 power savings of up to 35% can be reached by our optimization model in both topologies. Moreover, the heuristic approach allows to obtain close-to-optimal power savings with differences under 4% (Nobel-US) and 8% (Geant). It is also observed in both networks that lower savings are achieved when the percentage of transit nodes decrease from 50% to 10%. This behavior is expected given that with fewer transit nodes a smaller number of forwarding devices can be put into sleep mode, which yield the mayor contribution to the attained power savings. Additionally, with fewer transit nodes a higher number of demands are handled, thus more paths need to be established to accommodated such traffic. On the other hand, increasing the number of controllers implies in some cases a reduction in the power savings.

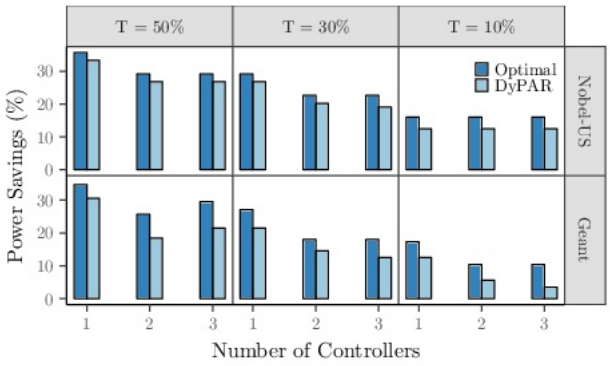


Fig. 2. Power savings in the Nobel-US and Geant topologies as a function of controllers amount, varying the percentage of transit nodes (T).

C. Assessment of Power Saving Potential

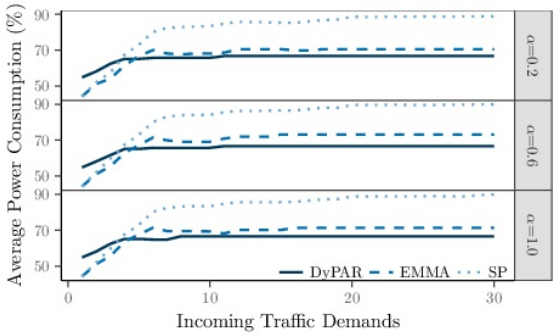
Due to the computational complexity of the exact model in networks similar in size or larger than Geant (see [15] for similar running time values), in what follows we use our heuristic algorithms. This is done taking into account a dynamic scenario with connection requests generated following the procedure previously explained. Several test were conducted and average values have been determined with a margin error less than 5.5% in the three considered networks, estimated by running our algorithm 10 times with different prime number seeds on each traffic configuration instance.

In terms of average running time of the algorithms, the off-line GrIS module requires around 39 ms (Nobel-US), 0.25 s (Geant) and 283 s (Cost266). Meanwhile, the DyPAR algorithm takes always less than 6.4 ms (Nobel-US), 16.5 ms (Geant) and 282.6 ms (Cost266), for all the considered traffic patterns. These values reveal the suitability of the proposed strategy for real-world deployments and its adequate scalability in terms of network size and traffic load. Due to space limitation, we may focus our attention on some specific traffic pattern configuration, but the general conclusions derived from performed evaluations hold for all the considered values.

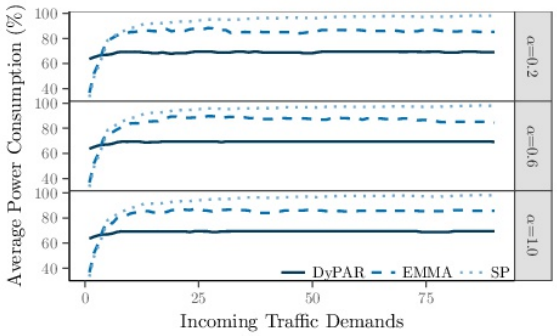
In addition, in order to evaluate the benefits of our proposal we compare the performance of the proposed algorithms with other two existing energy-aware routing approaches presented in related works [7] and [8], referred to here as SP and EMMA, respectively. As we are considering an in-band SDN, required control plane communications will be also established by these

two approaches. At the same time, shortest paths used by SP and EMMA are computed holding restrictions established to avoid additional traffic load through the network controller (i.e. data traffic cannot be routed through this device). On the other hand, we set the time threshold for demands reallocation (half of connection expected duration) and the number of transit nodes ($T = 50\%$) as in [8] for the three algorithms used in this comparison. Given the lack of support in SP and EMMA for network environments with multiple controllers we only consider the case of having one centralized network controller. However, the derived conclusions are general and a similar behavior is expected in case of having multiple controllers.

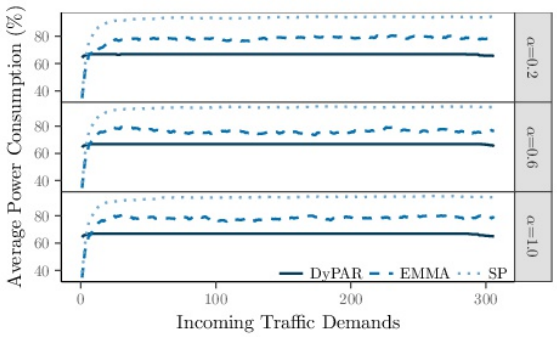
Fig. 3 shows the power consumption achieved by the three algorithms considering different topological scenarios and over-provisioning factor (α). These results correspond with an average arrival time of 0.2 demands/s and a mean holding time of 100 s, but similar values have been obtained for all the considered traffic patterns. Given the initial control plane configuration performed by the GrIS module, in the three considered topologies the other two methods exhibit a better behavior at the beginning of simulations. However, after allocating few demands more power can be saved by our approach. As it is shown, in terms of consumed power, DyPAR outperforms SP in all cases and it is generally better (in some cases just slightly better) than EMMA. For instance, after routing all incoming traffic, DyPAR attains power consumption reductions of up to 26.5% and 19.4% with respect to SP and EMMA, respectively. The reason is that SP only uses precomputed shortest paths to allocate the incoming traffic, while EMMA also performs a power-aware rerouting any time the active topology changes in order to find better paths for already allocated flows. On the other hand, power improvements achieved by our proposal are consequence of the combined GrIS/DyPAR operation where a minimum network subset is initially activated and new network elements (nodes and links) are only added when the incoming demand cannot be allocated on the currently active topology.



(a) Nobel-US topology.



(b) Geant topology.



(c) Cost266 topology.

Fig. 3. Power consumption in the three topologies with one controller as a function of traffic arrival, varying the over-provisioning factor (α).

D. Performance Degradation Avoidance

These power savings are only valid if the performance of QoS sensitive demands is not compromised. Moreover, to avoid overloaded networks a capacity reserve is typically set. So far, we had not considered this capacity margin, but now we analyze how the number of allocated demands is impacted when facing a more demanding traffic pattern and in presence of a MLU constraint. In this evaluation we set the average arrival time to 5 demands/s and the mean holding time to 200 s, while keeping the over-provisioning level equal to 1, since this represents the most demanding of the considered traffic patterns for the heuristics and the most critical from the performance degradation perspective.

Fig. 4 shows the percentage of demands that can be allocated by DyPAR, EMMA and SP in Nobel-US and Geant using different values of MLU. As it shown, DyPAR is able to reduce the blocking rate with respect to the other two approaches as a result of the CONGESTION_AWARE_REROUTING performed by this solution. In particular, while only negligible blocking rates are attained by our approach (less than 1.2%), up to 7 and 12 demands are blocked by SP and EMMA, respectively. SP performs better than EMMA given that in case of having more than one candidate route this algorithm selects the one leaving more available link capacity.

Intuitively, the capacity to successfully allocate the incoming traffic will not only be a result of the performed routing, since it is also related to the considered topology. In network topologies with more path redundancy a higher number of requests can potentially be accommodated. This difference can be noticed between Nobel-US and Geant, where an increase of allocated demands of up to 26.7% and 15.6% can be reached, respectively. Cost266 is not shown in Fig. 4, since a complete

(a) Nobel-US topology.

(b) Geant topology.

Fig. 4. Number of allocated demands with one controller as a function of traffic arrival, varying the MLU.

routing was always achieved in this topology by the three compared algorithms under the considered traffic patterns and MLU levels.

VII. Conclusion

In this paper we proposed a power-aware strategy that reduces the number of active nodes and links used to handle the incoming traffic suitable for SDN environments with in-band control traffic and multiple controllers. To achieve such goal, we first provided a link-based formulation of the optimization problem, integrating the routing requirements for data and control traffic. For large-scale topologies a heuristic approach is conceived combining a static control plane configuration with a dynamic power-aware routing. Besides being compatible with SDN environments without a dedicated control network, this strategy is able to handle demanding traffic arrival without degrading the performance of higher priority traffic. Through simulations using real-world topologies, we have validated that our heuristic approach allows to obtain close-to-optimal power savings, with differences under 8%. Furthermore, our proposal achieves better results in terms of power consumption and number of allocated demands than two existing related algorithms. For instance, after routing all incoming traffic, a reduction of power consumption of up to 26.5% and an increase of allocated demands of up to 26.7% can be reached by our solution. Lastly, it is important to emphasize that to exploit the reported benefits of our approach, fast switching-on technologies, allowing quick responses and low reconfiguration times between sleeping modes, are required for practical implementations. In the same way, additional criteria to ensure the capability of the network to quickly react in case of suddenly failures should be further analyzed. Therefore, the inclusion of restoration mechanisms in order to improve the fault tolerance capacity of our approach will be an important future task.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021," White Paper, Cisco, Sep. 2017.

[2] R. S. Tucker, "Energy Consumption in Telecommunications," in *Proc. Optical Interconnects Conference*, May 2012, pp. 1–2.

[3] M. Gupta and S. Singh, "Greening of the Internet," in *Proc. ACM SIGCOMM*, 2003, pp. 19–26.

[4] D. Kreutz, F. M. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[5] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Automatic Bootstrapping of OpenFlow Networks," in *Proc. IEEE LANMAN*, Apr. 2013, pp. 1–6.

[6] A. P. Bianzino, C. Chaudet, F. Larroca, D. Rossi, and J. L. Rougier, "Energy-Aware Routing: A Reality Check," in *Proc. IEEE GLOBECOM*, Dec. 2010, pp. 1422–1427.

[7] B. Özbek, Y. Aydoğmuş, A. Ulaş, B. Gorkemli, and K. Ulusoy, "Energy Aware Routing and Traffic Management for Software Defined Networks," in *Proc. IEEE NetSoft*, Jun. 2016, pp. 73–77.

[8] S. S. Tadesse, C. Casetti, C. F. Chiasserini, and G. Landi, "Energy-Efficient Traffic Allocation in SDN-basec Backhaul Networks: Theory and Implementation," in *Proc. IEEE CCNC*, Jan. 2017, pp. 209–215.

[9] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," in *Proc. USENIX NSDI*, 2010.

[10] F. A. Moghaddam and P. Grosso, "Linear Programming Approaches for Power Savings in Software-Defined Networks," in *Proc. IEEE NetSoft*, Jun. 2016, pp. 83–87.

[11] H. Zhu, X. Liao, C. de Laat, and P. Grosso, "Joint Flow Routing-Scheduling for Energy Efficient Software Defined Data Center Networks: A Prototype of Energy-Aware Network Management Platform," *Journal of Network and Computer Applications*, vol. 63, pp. 110–124, 2016.

[12] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A Power Benchmarking Framework for Network Devices," in *Proc. IFIP-TC*, 2009, pp. 795–808.

[13] A. Fernández-Fernández, C. Cervelló-Pastor, and L. Ochoa-Aday, "Achieving Energy Efficiency: An Energy-Aware Approach in SDN," in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–7.

[14] ——, "Energy-Aware Routing in Multiple Domains Software-Defined Networks," *ADCAIJ*, vol. 5, no. 3, pp. 13–19, Nov. 2016.

[15] ——, "Energy Efficiency and Network Performance: A Reality Check in SDN-Based 5G Systems," *Energies*, vol. 10, no. 12, Dec. 2017.

[16] F. Giroire, D. Mazauric, J. Moulierac, and B. Onfroy, "Minimizing Routing Energy Consumption: From Theoretical to Practical Results," in *Proc. IEEE/ACM GreenCom*, Dec. 2010, pp. 252–259.

[17] Gurobi Optimizer (Version 7.5). [Online]. Available: http://www.gurobi.com/

[18] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0-Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[19] L. Chiaraviglio, M. Mellia, and F. Neri, "Minimizing ISP Network Energy Cost: Formulation and Solutions," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 463–476, 2012.

[20] J. Li, J.-H. Yoo, and J. W.-K. Hong, "Dynamic Control Plane Management for Software-Defined Networks," *International Journal of Network Management*, vol. 26, no. 2, pp. 111–130, 2016.

Proactive Rerouting in Network Overlays

Reuven Cohen Yuval Dagan Gabi Nakibly
Dept. of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel

***Abstract*—Virtual overlay network technology provides important benefits to large data centers and to service providers. These benefits include traffic isolation and ease of service provisioning. When the underlying network supports traffic engineering, tunneling brings another important benefit: the ability to control the exact route of all packets without handling each independent flow. This paper addresses the problem of rerouting when the core network supports traffic engineering. We introduce the novel concept of proactive (time-driven) rerouting, which we distinguish from the well-known concept of reactive (event-driven) rerouting. One important advantage of proactive rerouting is reducing the communication between the core network controller and the edge network controller. Another advantage is that new flows do not have to wait before they are admitted into a rerouted tunnel. Unlike a reactive rerouting algorithm that knows which tunnel has to be rerouted, a proactive rerouting algorithm does not receive as an input the identity of a specific tunnel. Thus, its main goal is to predict which tunnel to reroute in order to increase the probability that future flows will be accommodated. Our main contribution is the development of a proactive rerouting algorithm that performs very well, sometimes even better than the reactive algorithms.**

I. Introduction

Software defined networking (SDN) and network virtualization use tunneling as a means of communication. Tunneling is used in the building of virtual overlay networks, which are known to better support virtual machine (VM) provisioning, to enable scalability and to improve automation. Virtual overlay network technology provides benefits to large data centers, the most important of which is traffic isolation for multi-tenancy. Another important benefit is ease of VM provisioning, because a VM can be migrated to a new subnet without changing its IP address or other network-dependent attributes. However, when the underlying network is based on a virtual circuit technology, such as MPLS, tunneling brings yet another important benefit: the ability to control the exact routes of the data packet, a process also known as traffic engineering, *without handling each independent flow*.

In a network overlay, also known as "overlay SDN," a packet is encapsulated inside another packet. The encapsulated packet is then forwarded along the route determined by the encapsulating header, until it reaches the end of tunnel, where it is de-encapsulated. Many different tunneling protocols are used today, including MPLS, VXLAN, NVGRE, STT and NVO3. The idea is that tunnels are built along routes with available bandwidth and flows are routed over these tunnels while taking advantage of the network resources reserved for their tunnels. The combination of SDN with MPLS has been identified by [2] as the best way to bring SDN into the carrier networks.

This research was partially funded by the Office of the Chief Scientist of the Israel Ministry of Economy under the Neptune generic research project. Neptune is the Israeli consortium for network programming.

ISBN 978-3-903176-08-9

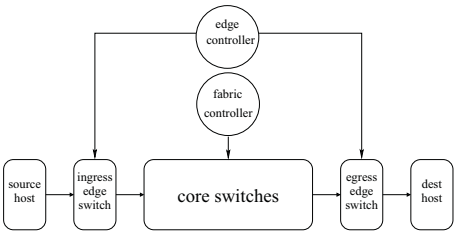


Fig. 1. The considered network architecture, as proposed by [2]

Figure 1 shows the three components of the considered network as described in [2]: hosts, edge switches, and the core fabric. The core and the edge are controlled by separate controllers: the edge controller handles the interface between the operator and the network, whereas the core controller is responsible for building tunnels and allocating resources to them.

The core network controller is responsible for creating new tunnels, each of which may contain thousands of flows at any given time. Once a tunnel is ready, the admission of new flows into it is the responsibility of the edge controller. These two controllers sometimes need to interact, as they do, for example, when the edge controller wants to admit a new flow but there is no tunnel with enough available bandwidth. The interaction between the two controllers can be a bottleneck in the network. We therefore seek to minimize it by requesting that the core controller maintain the tunnels without receiving explicit requests from the edge controller. This is the rationale behind the proactive algorithm presented later on.

Consider a pair of nodes (s, d). Let the tunnel between them be $t(s, d)$. A default tunnel is usually established over the shortest path with sufficient resources between s and d (i.e., the shortest path after ignoring the links without sufficient resources), but it can be established over any other path as well. When $t(s, d)$ runs out of resources, the edge controller cannot admit new flows into it. This controller communicates with the core network controller and requests that additional resources be allocated to this tunnel. If additional resources cannot be allocated, the only way to admit additional flows is either by building a new tunnel or by moving the tunnel to a new path, a process also known as *rerouting*. The advantage to rerouting of tunnels is that all the already admitted flows are

rerouted together with the tunnel, with no additional per-flow overhead [7].

Rerouting is preferable over creating additional tunnels for several reasons:

1) Every tunnel requires expensive forwarding entries in the network switches.
2) Every tunnel needs to be protected against failures [6].
3) With many tunnels between every pair of nodes, bandwidth utilization decreases due to a lower multiplexing ratio (a single 10Gb/s tunnel can be better utilized than 10 1Gb/s tunnels).

This paper addresses the problem of tunnel management and rerouting when the core network controller can determine the exact route over which every tunnel is established. Rerouting schemes have been extensively studied in the context of virtual circuit technologies, such as ATM, WDM, and MPLS. While most works on rerouting have focused on restoration following a link or node failure [7], [6], [19], [22], there is also extensive work on rerouting for throughput maximization[10], [27], [16], [25], [3], [14], [26]. The main difference between these works and the present work is that they all assume exact knowledge of the flows introduced into the network and of the flows that cannot be accommodated. Thus, they are all *reactive*. In contrast, the SDN core controller is not directly aware of the flows introduced into the network. Thus, it does not know about specific routing failure events. Such a controller can therefore use only proactive rerouting algorithms, which require no exact knowledge about how current flows are routed.

An important advantage of proactive rerouting is that *it requires no communication between the two controllers.* Another important advantage is that new flows do not have to wait for rerouting before they are admitted into a tunnel, because it alleviates congestion in hot-spots before the appearance of new flows.

Our key contribution is the distinction between reactive (event-driven) and proactive (time-driven) reroutings. With reactive rerouting, a tunnel may be rerouted by the core network controller only after the edge controller fails to admit a new flow into the network due to lack of bandwidth in the existing (default) tunnel between the corresponding end nodes. With proactive rerouting, the core controller is periodically invoked in order to replace the tunnels in the network such that the likelihood that the edge controller will fail to admit future data flows is minimized. We present algorithms for each model, in order to find the one that yields the best trade-off between the number of reroutings and the core network throughput.

The rest of the paper is organized as follows. Section III describes an algorithm for proactive rerouting. Section IV describes several algorithms for reactive rerouting. Section V presents simulation results for the various algorithms. Finally, Section VI concludes the paper.

II. Related Work

This paper is largely motivated by recent works that show the potential of combining SDN with MPLS. In [2], the authors suggest that a network fabric should be included as an architectural building block within SDN. They also identify the key properties for these fabrics: separation of forwarding and separation of control. They suggest that this separation would require an "edge" version of OpenFlow, which is much more general than the legacy OpenFlow, and a "core" version of OpenFlow, which resembles a slightly expanded version of MPLS label-based forwarding.

The benefit from using an overlay SDN is also discussed in [12]. The authors indicate that SDN has been successfully applied to data centers and campus networks but it has had little impact in the fixed wireline and mobile Telecom domain. They propose using "vertical forwarding" (tunneling) for extending SDN so that it can tackle the challenges of the Telecom domain. They also claim that tunneling enables flow-based policy enforcement, mobility and security.

Tunnel rerouting has been explored mainly in the context of virtual circuit technologies such as ATM [5], [7], MPLS [6] and WDM [21], [18]. It is usually used either when the original tunnel fails or does not have sufficient bandwidth for new flows. In [20], a "fast reroute" scheme is proposed in the context of MPLS. The main idea is to build a predefined bypass for each switch or link along the tunnel. When a switch learns that its upstream link or upstream neighbor on a given tunnel has failed, this switch can immediately forward the traffic along the pre-established bypass. The main advantage of this scheme is its very fast reaction to failures, thereby minimizing packet loss. However, this scheme is expensive from a management perspective, because it requires many bypasses for each tunnel, and from a bandwidth perspective, because bandwidth must be reserved in advance for each bypass.

In [6], the bandwidth cost of fast reroute was compared to the bandwidth cost of other rerouting schemes. This paper presents a comprehensive study of restorable throughput maximization in MPLS networks. One of its conclusions is that if the goal is to maximize revenue, fast reroute (referred to as "local recovery" in [6]) should be the recovery scheme of choice.

In [4], the authors study four rerouting algorithms to determine how the characteristics of the underlying network topology might affect their performance. They found that when the average node degree is small, most common practices for route placements, such as the shortest path algorithm, yield good performance in terms of the blocking ratio and that there is probably little advantage to rerouting. But when the average node degree increases, so does the number of available paths, and rerouting tends to improve the performance.

A recent line of research deals with rerouting of connections in elastic optical networks to alleviate bandwidth fragmentation [28], [24], [29]. Elastic optical networks allocate spectrum based on contiguous subcarrier slots with bandwidth. In such networks, dynamic setup and tear-down of connections can create bandwidth fragmentation, namely, non-contiguous slots that are not aligned along the routing paths and therefore cannot be used by new connections. In such networks, rerouting is needed to decrease the level of fragmentation and reduce the blocking probability of new requests. In [28], for example, a proactive rerouting scheme is proposed based on the state of the network links. However, rerouting in elastic optical networks have different constraints and objective function compared to the SDN case we deal with. In particular, in SDN

there is no bandwidth fragmentation.

Our paper focuses on tunnel rerouting due to lack of bandwidth over the original path. Therefore, the main theoretical problem is to find an alternative path with sufficient bandwidth. Sometimes rerouting a single tunnel will not suffice, and more tunnels need to be rerouted together. In such a case, the rerouting problem is computationally equivalent to the well-known NP-hard unsplittable multicommodity flow problem [9], [8], [11], [13], [15].

III. Proactive Rerouting

In proactive rerouting, the core controller is periodically invoked in order to reroute tunnels in a way that gives the edge controller more flexibility in accommodating future flows. The core network controller has no idea about future demands, not even their statistical distribution. We present the Network State Algorithm, which associates a "cost" with every link and takes the sum of the costs of all links as the network cost. The algorithm seeks to find a tunnel and a new path for this tunnel such that the network cost after rerouting the tunnel to the new path is minimized. The cost of every link reflects the load and the available bandwidth on it.

Let $G = (V, E)$ be a directed graph representing the considered core network. There is a default tunnel $t(s, d)$ that should accommodate the flows from s to d. Let F be a set of traffic flows to be admitted into the network one at a time, in an online fashion. When a flow is introduced, its termination time is unknown and future flows are also unknown.

A high level description of the algorithm is as follows. Note that each link has a cost attribute and a different weight attribute:

1) For every tunnel $t(s, d)$
   (a) Remove the tunnel from the network.
   (b) Assign a weight to every link. This weight is used only for finding a new shortest path for t. The weight of each link e is the cost of this link if t is routed over e minus the cost of the link if t is not routed over e.
   (c) Run a shortest path algorithm between s and d on the graph with the weights determined in the previous step.
2) Choose for rerouting the tunnel whose new route imposes a minimum total network cost.

We now explain how the cost of each link is determined. For every link e, define

- cap(e) as the link capacity (in Mb/s, say);
- used(e) as the capacity used by tunnels that are routed over e (also in Mb/s);
- load(e) as cap(e)/used(e).

The cost of link e depends on load(e) using the following relationship:

$$c(e) = \frac{1}{\theta}\left(\exp(\theta \cdot \text{load}(e)) - 1\right), \qquad (1)$$

where $\theta > 0$ is a free parameter that determines how quickly the cost increases with the load. Specifically, when θ is close to 0, the relationship is linear, and when θ increases, the cost growth is much faster, as depicted in Figure 2. The (-1) in Eq. 1 makes no algorithmic difference, because adding a constant to the cost does not change the identity of the selected reroute. It just guarantees that the cost equals zero if the link is unused.

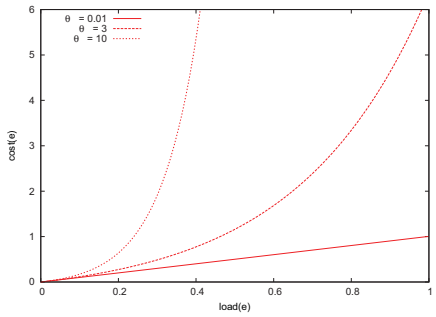


Fig. 2. The correlation between the cost and the load as a function of θ

Function *RerouteOne()*
  for *every tunnel* $t \in T$ **do**
    for *every link* $e \in E$ **do**
      if e *can accommodate* t **then**
        $c(e)_{+t} \leftarrow$ the cost of e if t is rerouted to a path that contains e
        $c(e)_{-t} \leftarrow$ the cost of e if t is rerouted to a path that does not contain e
        $w_t(e) \leftarrow c(e)_{+t} - c(e)_{-t}$
      else
        $w_t(e) \leftarrow \infty$
      end
    end
    $p_t \leftarrow$ the shortest path under the weight w_t
    $r_t \leftarrow$ the difference between weight of p_t and the weight of the old path of t, under the weight function w_t
  end
  Reroute the tunnel t^* into p_{t^*}, where t^* minimizes r_t over all tunnels t

Function *RerouteMany()*
  for $i = 1, \ldots,$ *num_reroutes* **do**
    RerouteOne()
  end

Fig. 3. A formal description of the Network State Algorithm (NSA)

Recall that the cost of the network is $\sum_{e \in E} c(e)$. Therefore, if θ is close to zero, the network cost equals the sum of the loads of all links in the network, whereas if θ is very big ($\theta > 1000$), the cost of the network is approximately equal to the cost of the most congested link.

Figure 3 gives a formal description of the algorithm. In this figure, $w_t(e)$ is the weight of link e considered by the shortest path in Step 1(c) of the informal description.

The cost function c gets a load level $\ell \in [0, 1]$ and returns the cost associated with this load. This can be expressed formally as follows:

$$c(\ell) = \frac{1}{\theta}\left(\exp(\theta\ell) - 1\right).$$

In the following discussion we use c interchangeably as a function that receives a link and as a function that receives a load.

The derivative $c'(\ell)$ indicates how costly it is to increase the load of a link whose current load is ℓ. Increasing the load from ℓ to $\ell + b$ increases the cost by

$$c(\ell + b) - c(\ell) = \int_{x=\ell}^{\ell+b} c'(\ell)du.$$

A calculation shows that $c'(\ell) = \exp(\theta\ell)$.

When θ is very small, the derivative is constant for any value of ℓ. Thus, the cost of increasing the load of a link from ℓ to $\ell+b$ is not affected by the link's current load ℓ. Therefore, the rerouting algorithm ignores the current loads on the links.

When θ is big, $c'(\ell)$ is much larger for big values of ℓ than for small values. Thus, it is much more costly to increase the load of an already congested link than it is to increase the load of a non-congested link. Generally, increasing the value of θ makes it more costly to increase the load of already congested links than of non-congested links.

Given a tunnel $t(s, d)$ whose bandwidth demand is demand(t), we now analyze the weight $w_t(e)$ assigned to e in order to find a shortest path from s to d when t is to be rerouted. We use the notations from the formal definition of the algorithm (Figure 3).

First, assume that θ is very small, namely, $c(\ell) \approx \ell$. For a link e, let load$(e)_{-t}$ be the load on e if t is rerouted to a path that does not contain e, and load$(e)_{+t}$ be the load on e if we reroute t to a path that contains e. It holds that

$$\begin{aligned} w_t(e) = c(e)_{+t} - c(e)_{-t} &= c(\text{load}(e)_{+t}) - c(\text{load}(e)_{-t}) \\ &\approx \text{load}(e)_{+t} - \text{load}(e)_{-t} = \frac{\text{demand}(t)}{\text{cap}(e)}. \end{aligned}$$

Thus, the shortest path for rerouting t is the path p from s to d for which

$$\sum_{e \in p} \frac{1}{\text{cap}(e)}$$

is minimized.

Next, assume that θ is very big; thus, for most values of $\ell_1 > l_2$ it holds that $c(\ell_1) >> c(\ell_2)$. The weight assigned to any link e is $c(e)_{+t} - c(e)_{-t} \approx c(e)_{+t}$. Thus, the shortest path for rerouting t is the path p from s to d for which

$$\sum_{e \in p} c(e)_{+t} \approx \max_{e \in p} c(e)_{+t} = \max_{e \in p} c\left(\text{load}(e)_{+t}\right)$$

is minimized.

The shortest path is the one that minimizes

$$\max_{e \in p} \text{load}(e)_{+t}.$$

Increasing the value of θ results in choosing longer rerouting paths, because a large number of low-load links can be added to the path without significantly affecting the cost. A simple example for this is given in Figure 4. In this figure, the numbers on the links indicate their capacities. Assume that the network has one active tunnel, from v_1 to v_3, whose bandwidth demand is 1. Assume that this tunnel is currently established over the path $v_1 \to v_3$. The controller is invoked for performing one reroute. The controller can actually leave the tunnel on its current route or move it to $v_1 \to v_2 \to v_3$. For $x = 3$, simple mathematical analysis reveals that if $\theta < 2.887$, the Network State Algorithm will leave the tunnel on its current path, and if $\theta > 2.887$ the algorithm will move the tunnel to the longer path $v_1 \to v_2 \to v_3$. For $x = 3.5$, the θ threshold decreases to ≈ 1.159, and for $x > 4$ the threshold is 0, namely, the algorithm always chooses the long path.

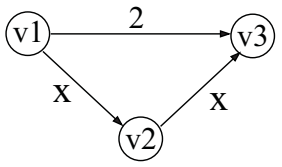


Fig. 4. An example of a network. The numbers indicate the capacity of each link.

IV. Reactive Rerouting

The purpose of this section is two-fold. First, we classify reactive rerouting into several schemes, depending on how much freedom is given to the controller during the rerouting process. Second, we present efficient and simple algorithm for each scheme. These algorithms are later used for comparing between the performance of reactive and proactive rerouting.

When the edge controller is unable to admit a new flow f into the network due to lack of bandwidth in the appropriate tunnel, it contacts the core network controller and requests to increase the bandwidth of the tunnel. If there is not enough spare bandwidth that can be added to the tunnel, the core network controller can invoke a rerouting algorithm. We will study the following rerouting schemes:

Scheme-A: The flow is rejected (rerouting is not performed). This scheme will be used as a benchmark.

Scheme-B: The default tunnel $t(s, d)$ of the considered flow, f, is rerouted to a new path that has sufficient bandwidth for all the existing flows that use this tunnel as well as for the new flow f.

Scheme-C: Any single tunnel from T can be rerouted in order to admit the new flow f, not necessarily $t(s, d)$.

Scheme-D: Up to N tunnels from T can be rerouted in order to admit the new flow.

Scheme-C is a generalization of Scheme-B, because for a given network state, every solution found by Scheme-B can also be found by Scheme-C. However, Scheme-C may reroute tunnels that are not related to the new flow, nor, in particular, to the same customer. Thus, not every network operator will prefer Scheme-C over Scheme-B. In the same way, Scheme-D is a generalization of Scheme-C.

A. Shortest-Path Algorithms

This subsection presents efficient (polynomial time) online algorithms for Scheme-B and Scheme-C, referred to as SP(B) and SP(C). For a given new flow, these algorithms are optimal in the sense that if a reroute exists, they will find it. For Scheme-D, an efficient algorithm that accommodates the new flow, if possible, while minimizing the number of rerouted tunnels, is unlikely to exist because this problem is NP-hard. This can be easily shown using a reduction to the well-known NP-hard unsplittable multicommodity flow problem [9], [8], [11], [13], [15]. Thus, the shortest path version for Scheme-D, referred to as SP(D), is only heuristic.

In all the following algorithms, let f' be a new flow that cannot be admitted into its default tunnel t'.

SP(B) Consider $G(V,E)$ while excluding the bandwidth used by all the tunnels except t' (none of this bandwidth can be used for the to-be-rerouted tunnel t'). From this graph, also excluded are the links whose available bandwidth is less than what is required to accommodate all the current flows of t' and the new flow f'. On the residual graph, if s and d are connected, the shortest path is chosen for t'. If s and d are not connected, f' is rejected because t' cannot be rerouted by Scheme-B.

SP(C) For each $t \in T$, where T is the set of tunnels, consider $G(V,E)$ while excluding the bandwidth used by all the tunnels except t. Try to accommodate the new flow f' along tunnel t'. If this is possible, try to accommodate t over the shortest path while considering only links whose residual bandwidth is sufficiently large. If f' cannot be accommodated or if t cannot be accommodated after f' is accommodated, repeat the procedure with another $t \in T$. If the procedure fails for all ts, reject f'.

SP(D) A similar procedure to $SP(C)$ is used. However, if a single reroute is insufficient, the tunnel whose rerouting maximizes the minimum available bandwidth over t' is chosen for rerouting. This can be stated formally as follows: for every $e \in E$, let avail(e) be the available bandwidth in link e. The tunnel chosen for rerouting is either t' or any other tunnel whose rerouting maximizes $\min_{e \in \text{path}(t')}$ avail(e). This procedure is repeated until flow f' can be admitted into t', but no more than N times. If f' cannot be admitted after N reroutings then no tunnel is actually rerouted. The pseudocode of this algorithm is as follows:

```
Run algorithm SP(C)
for i = 1, ..., N do
    if f' can be admitted into t' then
        stop
    end
    find the tunnel t ∈ T whose rerouting maximizes the
      minimum available bandwidth along t', and reroute
      it (the actual rerouting will be performed only if we
      succeed in admitting f')
end
No rerouting is performed and flow f' cannot be
  admitted into tunnel t'
```

B. A Linear Program Algorithm for Scheme-D

Since SP(D) does not guarantee an optimal solution to Scheme-D, we present here another algorithm for this scheme. This is an integer linear program algorithm, referred to as LP(D), whose main purpose is to serve as a benchmark for SP(D).

Let $F' \subset F$ be the set of flows admitted so far into the network. Recall that f' is a new flow that cannot be admitted into its default tunnel t'. The following LP parameters are defined:

- y_{te} – indicates whether tunnel t is currently routed over link e, $\forall t \in T$.
- b_t – denotes the bandwidth routed on tunnel t, while $b_{t'}$ already includes the demand of f'.

The following LP variables are defined:

- y'_{te} – indicates whether tunnel t will be routed over link e after the current iteration, $\forall t \in T$
- r_t – indicates whether tunnel t is rerouted.

The target function is to minimize $\sum_t r_t$, subject to several sets of constraints. The first set of constraints ensures flow conservation. The second set ensures that no link e carries more than its capacity cap(e). The third set ensures that the new path of each tunnel is identical to its old path unless this tunnel is rerouted. The fourth set ensures that at most N tunnels are rerouted, and the fifth set ensures that each flow is rerouted in only one path.

$$
\begin{array}{lll}
(1) & \sum_{e=(u,v)} y'_{te} - \sum_{e=(v,u)} y'_{te} & \\
& = \begin{cases} -1 & v = s \\ 1 & v = d \\ 0 & \text{else} \end{cases} & \\
& \forall v \in V, \forall t \in T, \text{where } s \text{ and } d & \\
& \text{are the tunnel source and destination.} & \\
(2) & \sum_t b_t \cdot y'_{te} \le \text{cap}(e) & \forall e \in E \\
(3) & y'_{te} + r_t \ge y_{te} & \forall \text{ tunnel } t \in T \\
(4) & \sum_t r_t \le N & \\
(5) & \sum_{e=(v,u)} y'_{te} = 1 & \forall t \in T, v = s \text{ (the source of } t) \\
(6) & y'_{te} \in \{0,1\} & \forall e \in E \ \forall t \in T \\
& r_t \in \{0,1\} & \forall t \in T
\end{array}
$$

V. Simulation Study

In Section V-A we evaluate the performance of the various reactive algorithms and in Section V-B we compare them to the proactive algorithm. Two criteria are relevant for this comparison:

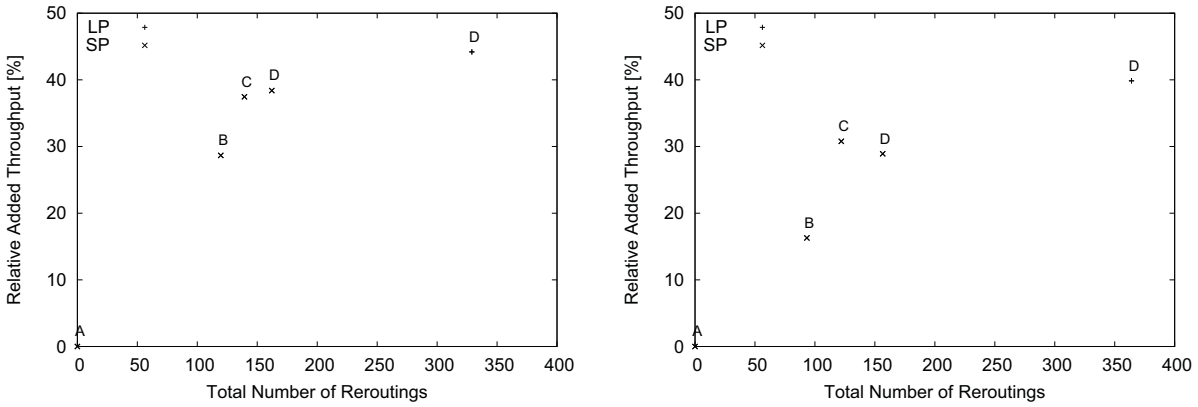
1) Relative added throughput, namely, how much greater the percent of throughput that each algorithm admits into the network compared to the case where no rerouting is allowed (Scheme-A). Formally, if a rerouting algorithm admits B Mb/s while only B' Mb/s is admitted without rerouting, then the relative added throughput for this algorithm is $(B - B')/B$. The relative added throughput is indicated in the y-axis of all the graphs presented in this section.
2) Management burden, defined as the total number of rerouting events. This number is indicated in the x-axis of all the graphs presented in this section.

Scheme-A gets no further mention because for this scheme the value of the y-axis and the value of the x-axis are always 0, by definition.

We expect to see some cases where the increase in relative added throughput is due to a substantial increase in the number of rerouting events. Thus, to get a good understanding on how each algorithm really performs with respect to the tradeoff between throughput and number of rerouting events, we always consider both criteria together.

A. Reactive Rerouting Simulations

We first study the performance of the reactive algorithms. Since SP(B) and SP(C) are optimal online algorithms for their schemes (when flows are received and considered one by



(a) Offered load = 0.5

(b) Offered load = 1

Fig. 5. The performance of the reactive algorithms SP(B), SP(C), SP(D) and LP(D) for small networks

one) while SP(D) is not necessarily optimal for its scheme, the LP(D) algorithm for Scheme-D is also simulated as a benchmark but, due to its computational complexity, this is done only for small-sized networks. Later, the results for larger networks will also be presented, but without LP(D).

Figure 5 shows the simulation results for the small scale networks. In this figure, 5 artificially generated topologies are considered, each with 15 nodes and 30 links. These topologies are generated using the BRITE simulator [1], which captures two important characteristics of network topologies: incremental growth and preferential connectivity of a new node to well-connected existing nodes. These characteristics yield a power law distribution to the degrees of the nodes. For each topology, 5 sequences of flows are generated, each with 1,000 requests. Each request is either for initiating a new flow or removing an existing flow. When a new flow is introduced, the algorithm is executed and the flow is either rejected or admitted following 0 or more rerouting events. The flow sequences are generated using the gravity model [17].

Figure 5 shows the simulation results for two levels of offered load: 0.5 and 1. The offered load of a network at a given time τ is defined as the sum of the offered loads of all the flows introduced to the network before τ (and that were accepted or rejected) whose termination time is after τ, divided by the total capacity of the network links. The offered load of a flow is the bandwidth demand of the flow multiplied by the number of links on the shortest path (while assigning a weight of 1 to every link) between the flow's source and destination. The flow sequences are generated for each simulation such that the network load remains roughly constant, and this yields one point on our graph, after it is averaged with additional executions using the same set of parameters but with a different seed.

It is evident from both graphs of Figure 5 that tunnel rerouting *significantly* increases the accommodated bandwidth. As expected, the schemes that accommodate more bandwidth impose a greater rerouting burden. In absolute numbers, more bandwidth is accommodated in heavily loaded networks. But the relative added throughput decreases when the load on the network increases because, in a heavily loaded network, there are fewer options for rerouting to gain additional throughput. This behavior is particularly noticeable for Scheme-B, because in this scheme only one tunnel can be rerouted.

For Scheme-D, the LP algorithm performs significantly more reroutings compared to SP(D), with only moderate increase in the admitted throughput. This is due to the fact that LP(D) is completely flexible in choosing the tunnels to reroute. In contrast, SP(D) is limited in the selection of a new path to rerouted tunnels.

To compare the performance of the various schemes, SP(B), SP(C) and SP(D), we define the "benefit-cost ratio" of an algorithm as the fraction obtained by dividing the relative added throughput by the number of reroutings, y/x. Using this ratio, SP(C) is the best algorithm, and SP(D) is better than LP(D).

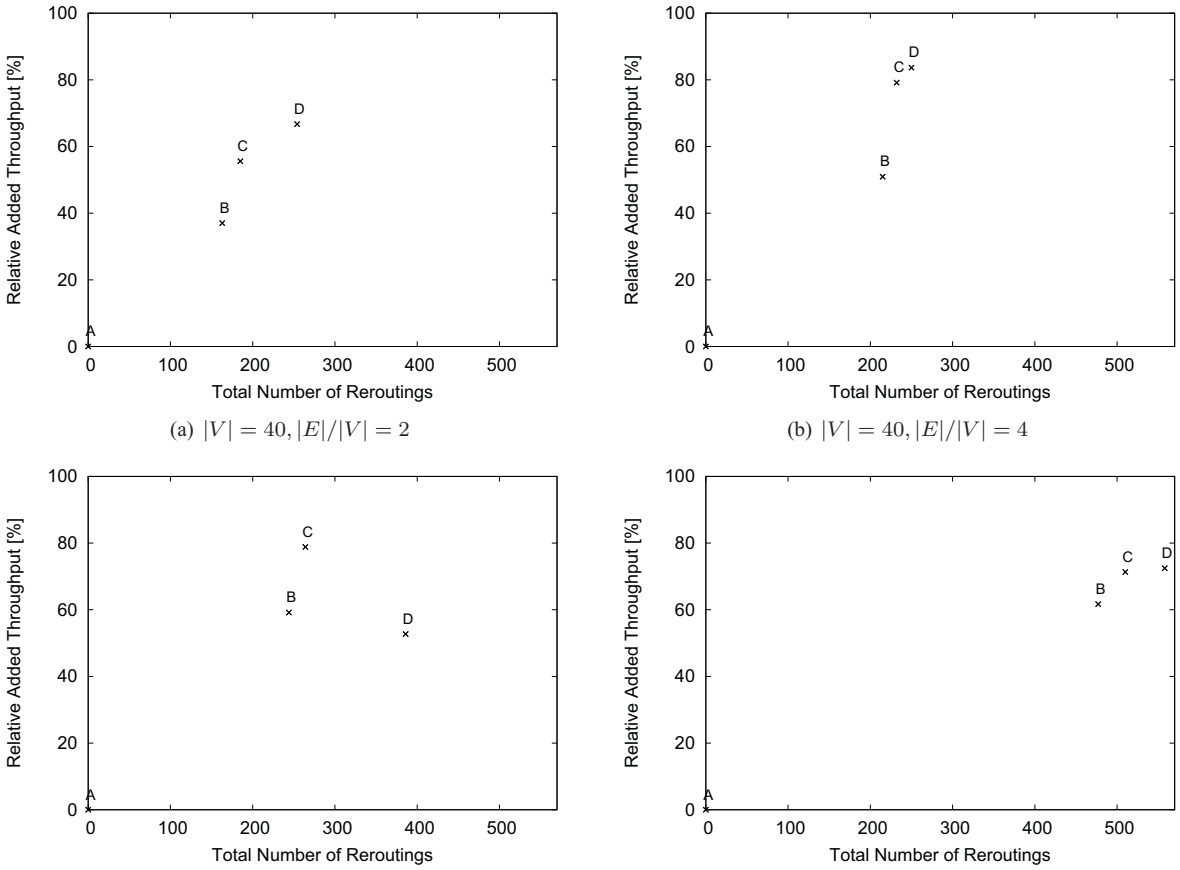
Figure 6 depicts the results for large networks. The networks simulated in Figure 6(a), (b) and (c) are synthetic networks built using BRITE with 40 nodes and 80 links, 40 nodes and 160 links, and 80 nodes and 320 links respectively. The network simulated in Figure 6(d) is a RocketFuel inferred topology[23] with 138 nodes and 730 links. LP(D) is not executed due to its exponential running time. For all these graphs, the offered load ratio is 0.5.

It is evident that the bandwidth accommodated by the algorithms, as well as the rerouting overhead, grow with the network size: as the network grows, there are more tunnel rerouting options. It is interesting to note that the various algorithms in the RocketFuel topology perform worse than they do for the synthetic topologies, despite the RocketFuel topology having more node links. But another property of this topology is that it has a few nodes with very large degree. These nodes create hotspots for which rerouting is less effective.

The results in Figure 6 strengthen our earlier finding that SP(C) yields a better benefit-cost ratio than SP(B) and SP(D). In other words, it results in a better tradeoff between the relative added throughput and the cost of rerouting.

B. Proactive Rerouting Simulations

This subsection compares the proactive approach and the reactive approach, by comparing the results of the three SP algorithms – SP(B), SP(C) and SP(D) – to those of the Network



(a) $|V| = 40, |E|/|V| = 2$

(b) $|V| = 40, |E|/|V| = 4$

(c) $|V| = 80, |E|/|V| = 4$

(d) RocketFuel Topology, $|V| = 138, |E|/|V| = 5.3$

Fig. 6. The performance of the reactive algorithms SP(B), SP(C) and SP(D) for large networks

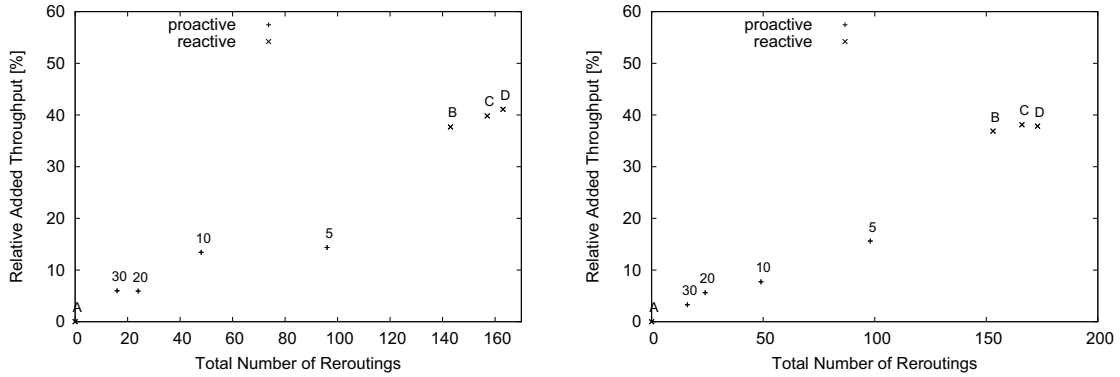
State Algorithm. Consider first Figure 7. The figure shows 4 graphs, for the same networks considered in the discussion of Figure 6. For each network type, the reactive SP algorithm is simulated for schemes B, C and D. Also simulated is the proactive Network State Algorithm for different execution periods: 5, 10, 20 and 40. The *execution period* is the average number of flows arriving between two consecutive invocations of the algorithm. Unless otherwise specified, for all these graphs the offered load is 0.25 and $\theta = 10$.

One cannot expect the proactive algorithm to admit as many flows as the reactive algorithms for a very simple reason: reactive algorithms are invoked only when the admission of a new flow fails, and the algorithm knows exactly which flow should be admitted over which tunnel. Thus, a reactive algorithm focuses on a specific tunnel and tries to increase the bandwidth available for it. In contrast, the proactive algorithm has no information about the flows that enter the network, and it is invoked regardless of whether past flows could or could not be accommodated.

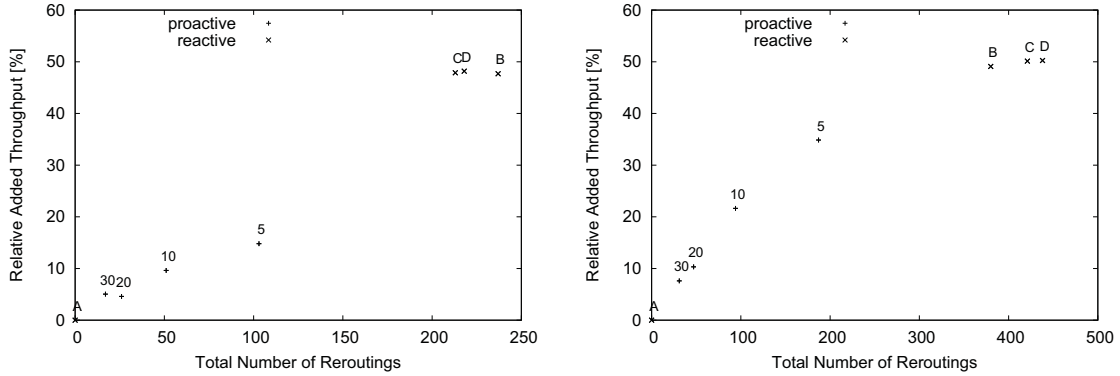
For each graph in Figure 7, consider first the four points that represent different execution periods for the Network State Algorithm (NSA). From these points we learn that determining the length of this interval is the most significant performance parameter. For example, in Figure 7(a) we see that when NSA is invoked every 5 time units it is able to admit into the network 15% more bandwidth while performing 95 reroutings. However, when it is invoked every 10 time units, it is able to admit the same percentage of extra bandwidth, but with an overhead of only 48 reroutings. This indicates that for this simulation instance NSA(10) performs much better than NSA(5). However, this is not always the case: in Figure 7(d) NSA(5) also performs many more reroutings than NSA(10), but it succeeds in accommodating much more bandwidth. This suggests that on small networks it is better to invoke NSA not very often, whereas in big networks NSA should be invoked more frequently.

When the NSA execution period is well chosen, the performance of NSA is surprisingly good compared to the performance of the best reactive algorithm. For example, consider Figure 7(d). In this figure, among all the reactive algorithms, SP(B) has the best benefit-cost ratio: $50/390 = 0.13$. However, for NSA(5) and NSA(10) the benefit-cost ratios are much better: $35/195 = 0.18$ and $21/100 = 0.2$ respectively! In Figure 7(c), the benefit-cost ratio of the best reactive algorithm, SP(C), is $48/220 = 0.22$, whereas the ratio of the best proactive algorithm, NSA(10), is $10/50 = 0.2$. We can conclude from this analysis that although the NSA algorithm cannot accommodate as much extra bandwidth as the reactive algorithms, it is competitive in terms of its benefit-cost ratio.

Figure 8 presents simulation results for the Network State Algorithm in two different Rocketfuel topology networks. In each network, the input traffic load is 0.25 and 0.5. The first

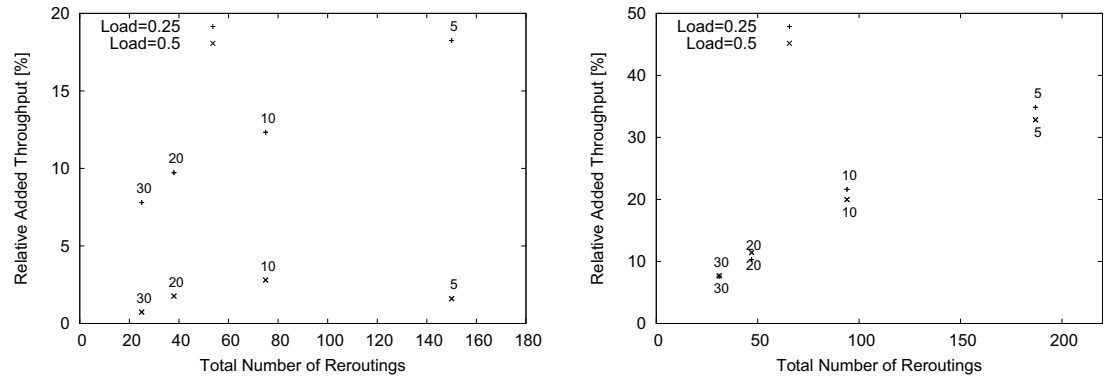


(a) $|V| = 40, |E|/|V| = 2$ (b) $|V| = 40, |E|/|V| = 4$



(c) $|V| = 80, |E|/|V| = 4$ (d) RocketFuel Topology, $|V| = 138, |E|/|V| = 5.3$

Fig. 7. The performance of the reactive algorithms vs. the performance of the proactive algorithm for various networks



(a) RocketFuel Topology $|V| = 80, |E|/|V| = 1.8$ (b) RocketFuel Topology $|V| = 138, |E|/|V| = 5.3$

Fig. 8. The performance of the Network State Algorithm for different offered loads

thing to note is that the offered load does not affect the number of reroutings. But this is expected because, in contrast to the reactive algorithms, the proactive algorithms are invoked periodically, regardless of the input traffic load.

When the benefit-cost ratio of the various algorithms is considered, it is evident that the performance of NSA for all execution periods is better on light loads. This is consistent with the results shown earlier for the reactive algorithms, and can be attributed again to the fact that a rerouting algorithm has more flexibility when the load is lighter. The simulation shown in Figure 8(a) resulted in much greater performance differences among the various algorithms as compared to the simulation shown in Figure 8(b). This is due to the much smaller link degree in the former, which makes effective rerouting more difficult. Figure 8(a) also shows that increasing the load has greater negative impact in this simulation.

Finally, Figure 9 shows the impact of θ on the simulation results for different execution periods. Each simulation is performed on 100 randomly created networks, using the same BRITE generator, each with 100 nodes whose average degree is 6. The x-axis denotes the execution period of NSA, namely, the average number of flows arriving to the network between

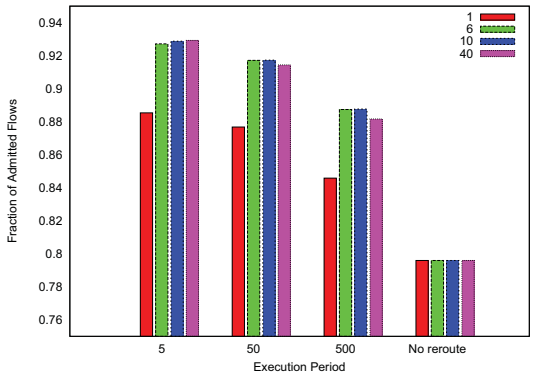


Fig. 9. The impact of θ on the performance of the Network State Algorithm

two invocations of the algorithm. The y-axis denotes the fraction of admitted flows. For each value of execution period there are 4 bars, denoting different θ values: 1, 6, 10 and 40. The leftmost bar for each period is $\theta = 1$. The y-axis denotes the total throughput. It is evident that θ has a significant impact on the performance. Moreover, large values of θ show significant improvement compared to small values. This can be explained by the fact that the initial routing of each tunnel is performed over the shortest path while ignoring the load on the various links. When rerouting is necessary, for high values of θ, the maximum load is kept low, and the network has room for new incoming flows. We can also see in this graph that **our proactive rerouting algorithm is very efficient even if it is invoked relatively rarely:** when the execution period is 500, namely, 500 new flows are introduced between two consecutive invocations of the algorithm, the percentage of rejected flows is reduced by 50%, from 0.2 (with no reroute) to only 0.1.

VI. Conclusions

This paper addressed the problem of tunnel rerouting in network overlays when the core network supports traffic engineering. For the first time, we introduced the concept of proactive (time-driven) rerouting, which we distinguish from the well-known concept of reactive (event-driven) rerouting. The main motivation behind proactive rerouting is to reduce the communication between the core network controller and the edge network controller, and to expedite the admission of new flows into the network.

We presented efficient algorithms for reactive rerouting, and then a novel Network State Algorithm for proactive rerouting. This algorithm associates a value with every network state, which indicates how well current tunnels take advantage of the available bandwidth resources. The algorithm tries to move tunnels such that the network state is maximally improved. Using simulations we show that although the Network State Algorithm cannot accommodate as many flows as the reactive algorithms, it performs surprisingly well with respect to the tradeoff between cost and value.

References

[1] I. M. A. Medina, A. Lakhina and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of MASCOTS*, 2001.

[2] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A retrospective on evolving SDN. In *HotSDN'12, Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[3] M. C. Chan and Y.-J. Lin. Behaviors and effectiveness of rerouting: A study. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 1, pages 218–223. IEEE, 2005.

[4] M. C. Chan and Y.-J. Lin. Behaviors and effectiveness of rerouting: a study. In *IEEE International Conference on Communications (ICC)*, volume 1, 2005.

[5] R. Cohen. Smooth intentional rerouting and its applications in ATM networks. In *Infocom'94*, June 1994.

[6] R. Cohen and G. Nakibly. Maximizing restorable throughput in MPLS networks. *IEEE/ACM Transactions on Networking*, 18(2), April 2010.

[7] R. Cohen and A. Segall. Connection management in ATM networks. In *Infocom'94*, June 1994.

[8] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.

[9] J. A. et al. Online load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44(3):486–504, 1997.

[10] V. Friesen, J. J. Harms, and J. Wong. Resource management with virtual paths in atm networks. *IEEE network*, 10(5):10–20, 1996.

[11] N. Garg and J. Koenemann. Faster and simpler algorithms for multi-commodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2), 2007.

[12] G. Hampel, M. Steiner, and T. Bu. Applying software-defined networking to the telecom domain. In *Infocom workshop*, 2013.

[13] J. T. Havill and W. Mao. Greedy online algorithms for routing permanent virtual circuits. *Networks*, 34:136–153, 1999.

[14] A. E. Helvaci, C. Cetinkaya, and M. B. Yildirim. Using rerouting to improve aggregate based resource allocation. 2008.

[15] S. G. Kollopoulus and C. Stein. Improved approximation algorithms for the unsplittable flow problems. In *Proceedings of FOCS*, pages 426–435, 1997.

[16] M. Koubàa and M. Gagnaire. Lightpath rerouting strategies in wdm all-optical networks under scheduled and random traffic. *Journal of Optical Communications and Networking*, 2(10):859–871, 2010.

[17] J. P. Kowalski and B. Warfield. Modelling traffic demand between nodes in a telecommunications network. In *in ATNAC95*, 1995.

[18] M. Liu, M. Tornatore, and B. Mukherjee. Survivable traffic grooming in elastic optical networksshared protection. *Journal of lightwave technology*, 31(6):903–909, 2013.

[19] G. Mohan and C. S. R. Murthy. A time optimal wavelength rerouting algorithm for dynamic traffic in wdm networks. *J. Lightwave Technol.*, 17(3), Mar 1999.

[20] P. Pan et al. Fast reroute extensions to RSVP-TE for LSP tunnels. IETF RFC 4090, May 2005.

[21] C. Rozic and G. Sasaki. Optical protection cost of ip fast reroute on a fully connected ip network over a wdm ring. In *National Fiber Optic Engineers Conference*, page JWA3. Optical Society of America, 2011.

[22] D. A. Schupke and R. Prinz. Performance of path protection and rerouting for WDM networks subject to dual failures. In *Optical Fiber Communication Conference*, 2003.

[23] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with RocketFuel. In *Proceedings of the ACM SIGCOMM*, August 2002.

[24] T. Takagi, H. Hasegawa, K.-i. Sato, Y. Sone, A. Hirano, and M. Jinno. Disruption minimized spectrum defragmentation in elastic optical path networks that adopt distance adaptive modulation. In *European Conference and Exposition on Optical Communications*, pages Mo–2. Optical Society of America, 2011.

[25] S.-W. Wang and C.-Y. Wen. Lightpath-level active rerouting algorithms in all-optical wdm networks with alternate routing and traffic grooming. In *Information Networking (ICOIN), 2012 International Conference on*, pages 42–46. IEEE, 2012.

[26] A. Wason and R. Kaler. Rerouting technique with dynamic traffic in wdm optical networks. *Optical Fiber Technology*, 16(1):50–54, 2010.

[27] E. W. Wong, A. K. Chan, and T.-S. P. Yum. Analysis of rerouting in circuit-switched networks. *IEEE/ACM Transactions on Networking (TON)*, 8(3):419–427, 2000.

[28] M. Zhang, W. Shi, L. Gong, W. Lu, and Z. Zhu. Bandwidth defragmentation in dynamic elastic optical networks with minimum traffic disruptions. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3894–3898. IEEE, 2013.

[29] M. Zhang, C. You, H. Jiang, and Z. Zhu. Dynamic and adaptive bandwidth defragmentation in spectrum-sliced elastic optical networks with time-varying traffic. *Journal of Lightwave Technology*, 32(5):1014–1023, 2014.

SWIFT: Bringing SDN-Based Flow Management to Commodity Wi-Fi Access Points

Seppo Hätönen*, Petri Savolainen*, Ashwin Rao*, Hannu Flinck† and Sasu Tarkoma*
*University of Helsinki, †Nokia Bell Labs

***Abstract*—Wi-Fi networks are largely served using over-the-counter commodity Wi-Fi routers, access points (APs), and possibly with wireless controllers. Existing approaches to bring the benefits of Software-defined Networking (SDN) to such Wi-Fi networks suffer from availability, hardware requirements, and scalability issues. For instance, these approaches do not support enterprise APs whose firmware cannot be modified. Furthermore, we observe that using SDN controllers for managing all the flows traversing these APs and routers requires more than just installing an SDN switch on these devices. In this paper, we present our solution called SWIFT: Software-defined Wi-Fi Flow Management, for bringing SDN-based flow management to Wi-Fi networks using commodity Wi-Fi APs, Wi-Fi routers, and Wi-Fi controllers. We also detail its benefits, shortcomings, and possible use cases. Specifically, our solution significantly lowers the barrier-to-entry for deploying and conducting research on software-defined Wi-Fi networks.**

***Index Terms*—Network Management, SDN, Wi-Fi.**

I. Introduction

Wi-Fi is becoming the communication medium of choice in our homes and offices. This has compelled manufacturers of televisions, home-entertainment systems, and other devices that traditionally used wires for connectivity, to support Wi-Fi. Paralleling the growth of Wi-Fi is the growing demand to programmatically compose and manage communication networks using Software-Defined Networking (SDN) principles. However, in spite of its growing presence and importance, Wi-Fi has received significantly less attention in the SDN community compared to its wired siblings.

The growing importance of Wi-Fi and SDN, and the limited Wi-Fi support in existing SDN solutions, highlights the importance of addressing the roadblocks in bringing the benefits of SDN to networks which use commodity Wi-Fi access points (APs). The key roadblock is that commodity Wi-Fi APs are typically Wi-Fi hubs/bridges that simply forward packets. These include both open-source solutions such as *hostapd* [1] and *OpenWrt* [2], and also proprietary APs used in enterprise or home networks. Furthermore, these APs cannot take intelligent forwarding decisions because they cannot implement the match/action rules used by SDN switches such as Open vSwitch (*OVS*) [3]. As detailed in §II-B, *simply installing an OpenFlow switch such as OVS on an AP is not enough for managing the network traffic flows traversing the AP*.

The seminal work on bringing the SDN to Wi-Fi networks was OpenRoads [4] which used protocols such as the Simple Network Management Protocol (SNMP) for managing the Wi-Fi APs. The insights from OpenRoads were leveraged by several solutions such as ÆtherFlow [5], BeHop [6], and OpenSDWN [7]. However, these solutions cannot be used as-is in existing Wi-Fi networks because they suffer from scalability, hardware, and availability issues (see §II). We therefore focus on *bringing SDN-based flow management to Wi-Fi networks built using over-the-counter commodity APs and controllers.*

In this paper, we present SWIFT, an architecture for bringing SDN-based flow management to existing Wi-Fi networks. Our architecture leverages on commonly available technologies: Client Isolation and SDN switches such as *OVS*. Client Isolation prevents the Wi-Fi clients associated with an AP from communicating with each other. This feature is supported by a wide range of enterprise and consumer APs.[1] We bring SDN functionality to Wi-Fi networks by leveraging on Client Isolation to take control of all flows in the Wi-Fi network. The Intelligent AP technique achieves this by empowering devices running *OpenWrt* with *OVS*. In contrast, the Thin AP technique allows Wi-Fi APs that support Client Isolation to offload the flow management to external SDN switches.

Our key contributions are as follows.

- We enable existing SDN controllers to manage the network traffic flows in Wi-Fi networks built using commodity APs and routers. This significantly lowers the barrier-to-entry to deploy and experiment on software-defined Wi-Fi networks.
- Our Thin AP technique offloads the management of flows traversing APs to external SDN switches. This technique can be used in Wi-Fi networks and testbeds which use APs that cannot run an SDN switch such as *OVS* within the AP. For example, enterprise APs typically do not allow installation of custom firmware such as *OpenWrt*, and custom software such as *OVS*. Similarly, legacy commodity APs may not have the resources for running SDN switches.
- Our Intelligent AP technique integrates *OVS* with *OpenWrt*, and leverages the computational power of modern APs for implementing the forwarding decisions mandated by the SDN controller. This enables SDN controllers to manage the traffic flows at the edge of Wi-Fi networks.

Roadmap. In §II, we discuss related works and our motivation. We then detail our two techniques and their use cases in §III, and present the results of experimental evaluation of our techniques in §IV. We finally conclude in §V.

[1]Client Isolation has many aliases such as Wireless Isolation, AP or Station Isolation, Peer-to-Peer Blocking *etc.* In this paper we use Client Isolation.

ISBN 978-3-903176-08-9

TABLE I
COMPARISON TO THE STATE OF THE ART

Metric	CAPWAP	OpenRoads	ÆtherFlow	BeHop	OpenSDWN	SWIFT
Association control at controller	✓	-	-	✓	✓	✓
Association control at AP	-	✓	✓	-	-	✓
Configure AP using OpenFlow	-	-	✓	✓	-	-
Configure AP using other protocols	✓	✓	-	-	✓	✓
Manage and control commodity APs	-	✓	✓	✓	✓	✔
Manage and control enterprise APs	✓	-	-	-	-	✔
Packet forwarding at the controller	✓	-	-	-	-	-
Packet forwarding at the AP	✓	✓	✓	✓	✓	✓

Existing approaches come with a trade-off of scalability versus the ability to manage the flows between clients associated to the same AP. A ✔ implies that SWIFT allows AP management tools including enterprise AP management tools such as the Cisco Wireless LAN Controller to manage the APs.

II. Background and Motivation

In this section, we first present other proposed approaches for bringing SDN functionality to Wi-Fi networks. We then motivate our work by discussing the roadblocks preventing these approaches from being used in Wi-Fi networks using commodity APs and routers.[2]

A. Existing approaches to integrate Wi-Fi networks with SDNs

In Table I, we compare the existing approaches for bringing SDN functionality to Wi-Fi networks, namely, a) the Control and Provisioning of Wireless Access Points (CAPWAP) protocol [8], [9], b) OpenRoads [4], c) ÆtherFlow [5], d) BeHop [6], and e) OpenSDWN [7].

The CAPWAP protocol is one of the seminal techniques for managing Wi-Fi APs. While CAPWAP predates SDN, they share the same underlying principles: a logically centralized CAPWAP controller manages the APs and takes decisions based on the network state. The specifications of the CAPWAP protocol are fairly detailed, and its proprietary siblings from Cisco [10], Aruba Networks [11] and Ubiquiti Networks [12] manage vendor-specific Wi-Fi hardware. In spite of its limited support in open source solutions such as *OpenWrt*, CAPWAP serves as a cornerstone for *Software-defined Wi-Fi networks.*

OpenRoads, ÆtherFlow, BeHop, and OpenSDWN, primarily differ on the techniques used for managing client associations and client mobility. Client association can be handled either at the AP or at the controller. While handling client associations at the AP is easy to implement, a controller handling associations can be extended to manage hand-overs for implementing seamless connectivity. OpenRoads and ÆtherFlow handle client associations at the AP, while OpenSDWN and BeHop handle associations at the controller.

OpenSDWN and BeHop support client mobility by creating virtual APs (VAP) for the Wi-Fi clients. The Wi-Fi interface of APs typically support multiple networks (BSSID), which are extended as VAPs. OpenSDWN creates a unique VAP for each client, and during client mobility the controller migrates this VAP from one physical AP to another AP. BeHop uses a similar approach, where a single VAP can serve multiple clients. BeHop can also allow clients to locally select the best physical AP. However, OpenSDWN and BeHop do not scale as commodity APs may limit the number of VAPs that can simultaneously run on a physical AP; *e.g.* a Cisco 3700 AP supports 16 networks, *i.e.* 16 VAPs [13]. Furthermore, running multiple VAPs incurs significant performance overheads due to beacon frames [14]. This is a serious shortcoming given the increasing number of Wi-Fi devices at home and at work [15].

Each approach presented in Table I uses a different technique for managing APs. While OpenRoads uses SNMP, ÆtherFlow and BeHop extend OpenFlow to include commands to manage APs. ÆtherFlow uses a modified CPqD [16] OpenFlow switch to change AP configuration while BeHop uses a local agent. Similarly, OpenSDWN uses an agent at the AP that exposes configuration hooks to the controller.

B. Shortcomings of existing approaches

The existing approaches have limitations when dealing with flows between clients associated to the same AP. OpenRoads, ÆtherFlow, BeHop, and OpenSDWN are all built on top of *OpenWrt*, which in turn uses the Linux IEEE mac80211 driver [17]. This driver maintains a list of clients associated with the AP, which is used to directly forward packets between associated clients; packets between clients do not traverse the networking stack of the AP. A consequence of this optimization is that *an SDN switch such as OVS or CPqD running on an AP is unable to manage the flows between Wi-Fi clients associated with the same AP*. OpenSDWN and BeHop address this issue by creating VAPs. However, many APs support only a limited number of VAPs, limiting the number of clients served by a physical AP. Multiple VAPs, *i.e.* SSIDs, also incur heavy overheads in the Wi-Fi due to beacon frames, making them unsuitable for dense Wi-Fi networks or locations with multiple overlapping networks [14]; the current Wi-Fi design principles set the maximum number of SSIDs to only four.

The existing SDN approaches are also not suitable for enterprise APs such as those from Cisco. Typically enterprise APs neither offer support for SDN controllers nor do they support customization by third parties. While these APs can be configured to operate with a proprietary Wi-Fi controller or work autonomously, the limited customization support currently makes them impractical for many SDN research activities or integrating them to SDN-based networks.

Similarly, the existing approaches are also not suitable for legacy APs with limited storage space and limited computational capacity. The match-action rules mandated by SDN are computationally expensive compared to the direct packet forwarding. The legacy APs typically also have limited storage space (in the order of a few MB [18]) which might not be sufficient to add the binaries of SDN switches.

[2]Please note that, in this paper we use the terms Wi-Fi router and Wi-Fi AP interchangeably. A Wi-Fi AP typically acts as a bridge between Wi-Fi and wired networks. In contrast, Wi-Fi routers also include routing capabilities.

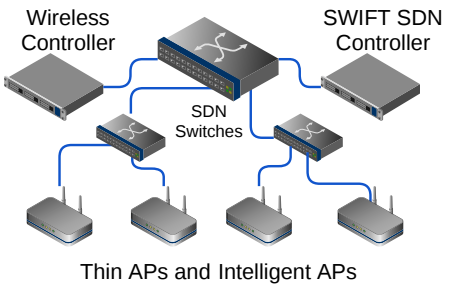


Fig. 1. **Example SWIFT topology**. *The SWIFT SDN controller manages the flows traversing the network, while the Wireless Controller manages the APs.*

C. Motivation

Existing approaches suffer from scalability, hardware, and availability issues for bringing the benefits of SDN-based traffic management to Wi-Fi networks built using commodity APs. This motivates us to find a solution which achieves this requiring only minimal changes to the APs and can be deployed on existing networks.

Programmatically managing a Wi-Fi network involves a) managing and provisioning the APs and b) managing the flows traversing these APs. The AP management and provisioning includes managing the radio interface parameters such as signal strength, channel, *etc.* This can be done either through existing wireless LAN controllers such as Cisco Wireless Lan Controller (WLC) [10], or through other network management systems. These are solid and mature solutions for managing commodity and enterprise APs. At the same time, SDN switches such as *OVS* [3] and SDN controllers[3] such as OpenDaylight [19] and Ryu [20] are solid and mature solutions for flow management. However, the existing approaches cannot combine SDN solutions with existing Wi-Fi networks built using commodity and enterprise APs. In particular, the SDN controllers cannot manage the flows between Wi-Fi clients associated with the same AP, while existing Wi-Fi APs do not support SDN.

In the following, we present our SWIFT architecture for bringing SDN-based flow management to existing Wi-Fi networks. Our solution enables existing SDN controllers to manage all the Wi-Fi traffic flows, including the flows between clients associated with the same AP, while allowing existing Wi-Fi controllers such as WLC to continue to manage the client associations, the radio interfaces, *etc.*, of these APs.

III. SWIFT: Software-defined Wi-Fi Flow Management

In this section, we present SWIFT, our software-defined Wi-Fi flow management architecture. The SWIFT architecture is illustrated in Figure 1, which depicts an enterprise network with a Wi-Fi controller. The Wi-Fi controller is used to manage the existing commodity APs, while the SWIFT controller manages the network traffic flows.

[3]In the rest of the paper we use the term SDN controller to refer to SDN-based network and flow management systems such as OpenDaylight and Ryu.

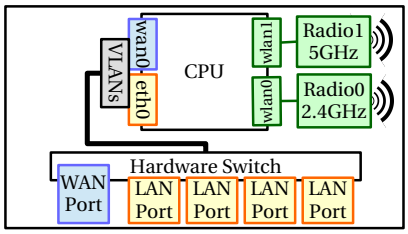


Fig. 2. **A Wi-Fi router that internally uses VLANs**. *Using VLANs to separate WAN and LAN ports reduces costs as only one physical interfaces is needed at the CPU.*

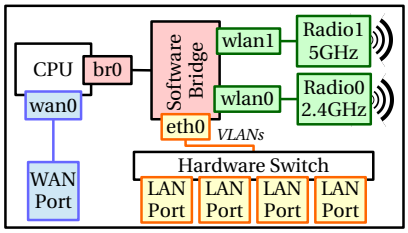


Fig. 3. **Interfaces on an *OpenWrt* router**. *Regardless of the internal wiring, OpenWrt uses a* Software bridge *to manage the Wi-Fi network and the LAN.*

A key building block for SWIFT is Client Isolation, a feature for preventing Wi-Fi clients from communicating with other clients associated to the same AP. When Client Isolation is enabled on an AP, the AP stops bridging the traffic between Wi-Fi clients associated with that AP. In III-A, we discuss Client Isolation and present ways in which it can be used to enable SDN switches to manage the Wi-Fi network traffic flows. We then detail the following two techniques for bringing SDN-based flow management to Wi-Fi networks.

a) Intelligent AP: In this technique (see §III-B), we run *OVS* inside the AP. This enables APs to exert fine-grained control over flows traversing its communication interfaces.

b) Thin AP: In this technique (see §III-C), an AP offloads the flow management to a remote SDN switch and in essence becomes a remote Wi-Fi interface on this switch; the AP and the SDN switch form the Thin AP.

In III-D, we discuss the steps the SWIFT SDN controller needs to take to manage flows traversing APs implementing our techniques, and why simply adding *OVS* to an AP is not enough to bring SDN to Wi-Fi networks.

A. Client Isolation

We now use the internals of *OpenWrt* running on commodity Wi-Fi routers to present an overview of Client Isolation.

In Figure 2, we present the connectivity between the interfaces of widely used commodity Wi-Fi routers. A typical Wi-Fi router has a WAN interface for the Internet connectivity, an Ethernet switch for the wired LAN ports, and at least one Wi-Fi interface. These interfaces are typically exposed to the CPU using Virtual LANs (VLANs). In Figure 2 we present a router (Netgear WNDR4300v1) that internally uses two VLANs, one VLAN for the WAN and another VLAN for the LAN. As shown in Figure 3, *OpenWrt* abstracts these wiring internals and exposes a software bridge that connects

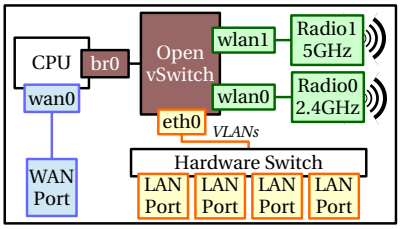


Fig. 4. **AP configured for the Intelligent AP**. *OVS replaces the default bridge provided by OpenWrt, and Client Isolation is enabled on the AP. This allows the OVS to manage the flows traversing the AP.*

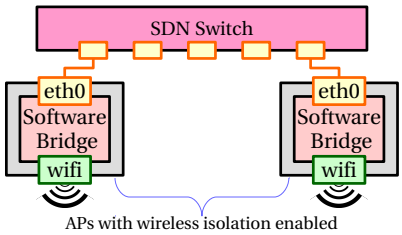


Fig. 5. **APs configured for the Thin AP**. *The APs have Client Isolation enabled, and an external SDN switch manages the network traffic flows traversing the AP.*

the Wi-Fi and wired LAN interfaces. However, because of the internal optimizations in the wireless LAN driver discussed in §II-B, the packets exchanged between the clients associated with the same Wi-Fi interface do not traverse this bridge.

This can be mitigated by enabling Client Isolation. However, we have observed the three following implementations in enterprise and consumer APs [2][10].

a) Permissive Isolation: The driver sends all traffic flows to the AP's network stack. An SDN switch running on the AP can therefore be used to manage the flows traversing the AP.

b) Restrictive Isolation: The driver only allows Address Resolution Protocol (ARP) messages to reach the network. An external SDN switch connected to the AP can use these ARP messages to impersonate the other hosts in the network. This can be achieved by using various techniques such as a) Proxy ARP for Private VLANs (PVLAN), also known as VLAN Aggregation [21][22], or b) the SDN controller sending an ARP reply with the MAC address of the switch in response to ARP requests from clients associated with the AP.

c) Total Isolation: The driver discards all traffic between wireless clients, which makes it impossible to extend Client Isolation to allow SDN switches to manage the traffic flows.

OpenWrt-based APs use *Permissive Isolation*, and enterprise APs may use any of the above implementations; the supporting documentation may provide hints on the implementation used by a given AP. For example, the Cisco 1131ag AP and Cisco WLC can be configured to either use *Restrictive Isolation* or *Total Isolation* [10]. In the following, we present two techniques to combine SDN switches with APs that implement either *Permissive Isolation* or *Restrictive Isolation*.

B. Intelligent AP

In this technique we run *OVS* on an *OpenWrt*-based AP. As shown in Figure 4, we replace the Linux Bridge created by *OpenWrt* (see Figure 3) with *OVS*. All interfaces that were plugged to the bridge are moved to the *OVS*, and Client Isolation is enabled on the Wi-Fi interfaces. Client Isolation forwards all packets arriving on these interfaces to the *OVS*. This allows the *OVS* to manage all flows between Wi-Fi clients and the flows traversing the AP to the wired network.

The Intelligent AP technique also supports multiple Wi-Fi networks (SSID) on the same AP. Each SSID appears as a logical Wi-Fi interface on *OpenWrt*, which is then plugged to the *OVS*. To manage the flows between Wi-Fi clients in different SSIDs, the SDN controller only needs to know the *OVS* port corresponding to a given SSID. The key benefit of this technique is the implementation of SDN match/action rules at the edge of the Wi-Fi network.

C. Thin AP

This technique is suitable for APs falling into one or more of the following categories.

- A custom firmware such as *OpenWrt* cannot be installed on the AP. For example, the flash memory size of the AP is not large enough to install *OpenWrt*.
- A custom software such as *OVS* cannot be installed on the AP. For example, the AP runs proprietary firmware which does not allow customization of the AP.
- The hardware restrictions make it impractical to implement the Intelligent AP approach.

Most enterprise APs fall into one or more of the above categories as they may not allow installation of third-party firmwares or software. Similarly, many legacy APs which support *OpenWrt* but have either limited storage or computational capacity to run *OVS* can use our Thin AP technique.

In this technique, the AP acts as a remote Wi-Fi interface for an SDN switch, and each Wi-Fi network of the AP becomes a port on that switch. This port and the AP connected to it form the Thin AP, which is now responsible for the flows traversing the AP. Furthermore, multiple APs can be connected to a single SDN switch, *i.e.* only a single SDN switch is required to turn a small Wi-Fi network into an SDN-managed one.

As discussed in §III-A, if the Client Isolation on an AP is of the type *Restrictive Isolation* then the SDN controller will be required to provide ARP responses to ARP queries made by clients associated with the AP. However, this impersonation may cause issues with device discovery, for example. We discuss these issues in Section §III-F. In contrast, Intelligent APs do not require the controller to handle ARPs because these APs do not redirect traffic to an external SDN switch.

The key benefit of the Thin AP technique is that it only requires Client Isolation on the AP. This enables the transformation of existing Wi-Fi networks to support SDN.

D. Flow Management using an SDN Controller

We now discuss the steps an SDN controller such as OpenDaylight or Ryu must take to manage flows traversing APs configured as either Intelligent AP or Thin AP.

1) Managing flows traversing Intelligent APs: The SDN controller must perform the following additional tasks to manage the flows traversing Intelligent APs. First, the controller must know which SDN switches are APs configured as Intelligent APs. Second, the controller also has to keep track of the hosts in the network that are associated with these APs, which is essential for forwarding packets between wireless clients associated with the same AP. For such flows, the packets received from the wireless interface of an Intelligent AP must be sent back to the wireless interface. The client's MAC address can be used for this. Additionally, for an Intelligent AP to support encrypted Wi-Fi traffic, packets with *EtherType 0x888e (EAP over LAN)* must be sent to the AP's network stack to be processed by the AP's WPA2 module.

2) Managing flows traversing Thin APs: A Thin AP has a few more requirements from the SDN controller than an Intelligent AP. In addition to Intelligent AP requirements, the SDN controller has to track the IP addresses of all the clients associated with each Thin AP. The SDN controller is expected to examine the ARP queries made by Wi-Fi clients for discovering the other wireless clients associated with the same Thin AP. If the clients are allowed to communicate with each other, the controller responds with an ARP reply with the MAC address of the SDN switch connected to the AP, for which the IP tracking is required. As a consequence of this, a Wi-Fi client α communicating with Wi-Fi client β will send a packet with the source MAC address of client α, source IP address of client α, destination MAC address of the SDN switch, and destination IP address of client β. For such packets the SDN controller commands the SDN switch to replace the source MAC address with the MAC address of the SDN switch, and replace the destination MAC address with the MAC address of client β, and finally send the packet to the wireless interface of the Thin AP. This allows the SDN switch to forward packets between clients associated with the Thin AP. Unlike the Intelligent AP, the Thin AP does not require any specific rules to support Wi-Fi encryption because these frames are managed locally by the AP or by a Wi-Fi controller.

3) Supporting Client Mobility: Most Wi-Fi controllers offer support for client mobility [10]. For the SDN controller to support mobility, the controller must update the flow table rules in SDN switches to ensure that packets are forwarded to the AP which the client is associated to. Both Wi-Fi and the SDN controllers can coordinate this using their respective north-bound APIs or react to network changes.

The above actions are straightforward to implement in any SDN controller, and we have implemented them in our SWIFT SDN controller based on the Ryu controller framework.

E. Use Cases

We now discuss some of the use cases of our work.

1) Bringing SDN-based flow management to enterprise Wi-Fi networks: Current enterprise APs are typically managed using a proprietary Wi-Fi management solutions and do not allow installation of custom firmwares. If these APs support either *Permissive Isolation* or *Restrictive Isolation*, and if SDN switches are added to the network, the Wi-Fi traffic flows can be controlled by SWIFT. With the steps discussed in §III-D, the existing Wi-Fi controllers can now function alongside SWIFT and manage both Wi-Fi APs and traffic flows.

2) Using existing Wi-Fi testbeds for SDN research: Many Wi-Fi testbeds contain a large number of legacy devices. The hardware in these devices might be obsolete by modern standards; for example, APs may have only 4MB of flash storage [18]. However, these devices can run *OpenWrt*, and our Intelligent AP or Thin AP techniques can be used on these devices as *OpenWrt* supports *Permissive Isolation*. This allows conducting SDN research in existing Wi-Fi testbeds.

3) Programmable Network-wide access control: An SDN controller with a fine-grained view of the devices in the network allows the implementation of network-wide access control. The SDN controller can be used to dynamically grant or deny devices access to network services and resources. This level of control has many use cases especially for Wi-Fi networks where many clients can enter and leave at will. First, guest devices can be granted only a limited Internet access while known devices have full access. Second, SDN controllers can coordinate with AP management controllers for creation of location specific access rules. By determining where each client is located, the SDN controller can manage which devices each client has access to. For example, access to streaming devices such as Chromecast or AppleTV can be restricted only to devices near the clients. Third, Wi-Fi networks are increasingly being used to connect Internet-of-Things (IoT) devices such as baby-monitors and security cameras, which are known to have vulnerabilities. With our techniques, these devices can be protected by creating an SDN-based security overlay which controls the set of devices with which such vulnerable devices can communicate.

4) Limit the number of SSIDs: Combined with the network-wide access control, the SWIFT can limit the number of SSIDs used to only a few while retaining the benefits of multiple SSIDs. Different security or group memberships can be allocated to Wi-Fi clients, allowing the dispensation of specific networks such as "guest" or "accounting".

F. Discussion

In this section, we presented our SWIFT architecture and the techniques it is built on. The techniques combine off-the-shelf components, mainly Client Isolation and SDN switches such as *OVS*, to provide SDN-based management of *all traffic* traversing a Wi-Fi network.

The Intelligent AP technique empowers routers and APs which are supported by open-source firmwares such as *OpenWrt*, and allow installation of an SDN switch such as *OVS*. The Thin AP technique can be used with existing Wi-Fi APs which do not allow custom firmwares, or are otherwise incapable of processing the computationally intensive SDN match-action rules. Our two techniques can be used *to bring SDN-based traffic management to existing Wi-Fi networks*.

Our techniques are straightforward to implement in any SDN controller. The Intelligent and Thin AP approaches have

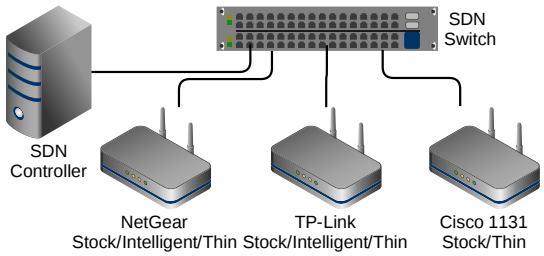


Fig. 6. **Testbed topology**. *The Netgear and TP-Link APs can be configured as Stock, Intelligent, or Thin APs, while the Cisco AP can be configured as Stock or Thin AP. We built our custom SDN controller using the Ryu framework.*

some specific needs that must be addressed. For example, when an AP is configured as a Thin AP, and a client sends out a broadcast packet, the SDN switch to which the AP is connected, sends broadcast packets back to the AP. This may trigger a loop detection algorithm on the AP, causing the AP to momentarily drop all packets sent back to it. This can either be disabled in the AP, or it can be circumvented with application-specific controller modules. For example, DHCP requests can be routed directly to a DHCP server.

Our two techniques can also be combined with the other solutions discussed in Table I. This would be beneficial as we currently lack the remote AP configuration capabilities, offered by for example Cisco WLC. At the same time, our techniques address the scalability issue of multiple networks on a radio interface; this is a serious shortcoming of existing solutions.

IV. Evaluation

In this section we present results of experiments conducted to a) quantify the overheads incurred by APs implementing our two techniques, and b) address scalability. Please note that the results presented here are for qualitative purposes only.

A. Experiment Setup for Quantifying Overheads of SWIFT

1) Devices Used and Network Topology: We used three commodity APs for our experiments: a) Netgear WNDR-4300v1, b) TP-Link WR1043NDv2, and c) Cisco 1131ag. The Netgear and TP-Link can run *OpenWrt* while the Cisco does not support custom firmware; each of these APs have different hardware capabilities. The Netgear and TP-Link can be configured either as a Stock, Intelligent, or Thin AP. We run these two APs in their Stock *OpenWrt* configuration for the baseline measurements. For the Intelligent AP tests, we install *OVS* on these APs, enable Client Isolation, and include a patch for *hostapd* to detect and exchange data with *OVS* to support WPA2 encryption. For the Thin AP tests, the default Linux bridge is used with Client Isolation enabled and the *OVS* is located in a remote host. The Cisco is used in its Stock configuration for baseline measurements, and Client Isolation is enabled only when the AP is configured as a Thin AP. The testbed topology is shown in Figure 6. We use a FIT-PC3 Pro as the main SDN switch for the testbed, and also as the external SDN switch for our Thin APs. In addition, we used two identical laptops as test clients.

2) SWIFT Controller: Our SWIFT controller extends Ryu SDN to support our Thin and Intelligent AP implementations. For each traffic flow, SWIFT installs corresponding flow rules after an SDN switch receives the flow; in our implementation, the forwarding decisions are based on the MAC addresses of the clients. These give a lower bound on overheads, while additional overheads can be incurred depending on the policies that determine the rules. Our main focus was on the overheads incurred by the redirection of packets to an SDN switch.

3) Test Scenarios: We use the following three scenarios. We believe that these three scenarios emulate small Wi-Fi networks and also testbeds used for Wi-Fi experiments.

Scenario A. In this scenario, our two laptops are associated with the same AP. This scenario emulates a small Wi-Fi network such as a home network with a single AP.

Scenario B. In this scenario, the two laptops are associated with different APs. This scenario emulates a Wi-Fi network in a small-office home-office scenario where the clients can be associated with different APs.

Scenario C. In this scenario, one laptop is associated with an AP, while the second laptop is in a high-speed wired network. This scenario emulates a network such as a university network where Wi-Fi clients communicate with servers present in a high-speed wired network. The client in the wired network is connected to the switch using a 1 Gbps Ethernet cable.

4) Traffic Generation: We use the *Flent network benchmarking tool* [23] for quantifying the overheads incurred by our changes to the APs. Our motivation for using Flent was that it includes a *Realtime Response Under Load (RRUL)* test for testing Bufferbloat [24]. During an RRUL test multiple TCP flows between our clients traverse our network at the same time. Furthermore, these competing flows can be configured to have different priorities. For different priorities, the Flent uses the Differentiated Services Code Point (DSCP) field in IP packets; all the packets of a given TCP flow have the same value in the DSCP field. This allows us to emulate the network traffic where clients use different applications such as VoIP and web browsing at the same time. This test is particularly important as *OpenWrt* internally uses the Linux networking stack where the default queue of the networking interfaces is `pfifo_fast`. In the `pfifo_fast` configuration, a queue has three bands labeled 0, 1, and 2, and the DSCP field in the IP header determines the band of a packet; packets in band 0 are served with a higher priority than those in band 1 which in turn have a higher priority than those in band 2 [25]. Each band is served First In First Out, and packets in a band are served only when there are no packets in a lower numbered band; for example, packets in band 2 are served only when there are no packets in band 0 and band 1. We use Flent's RRUL tests in the following three modes.

a) RRUL (default): In this mode, the Flent on each client creates multiple TCP and UDP flows with different priorities.

b) RRUL Best Effort: The Flent running on each client generates multiple TCP and UDP flows. However, in this test all traffic flows have the same priority, i.e., all TCP and UDP packets have the same value in the DSCP field in the IP header.

c) RRUL Ping: The Flent uses only ICMP messages to measure the round-trip time (RTT) without network load. This provides a baseline measurements on the impact of the Intelligent AP and Thin AP approach on the network latency.

We conduct the following experiments to quantify the overheads incurred by of our two techniques. For each of the three scenarios—Scenario A, Scenario B, and Scenario C—we run the Netgear and TP-link AP in the Stock, Intelligent AP, and Thin AP configuration, and the Cisco AP in the Stock and Thin AP configuration. Furthermore, we run 30 iterations of Flent in the default RRUL mode and the RRUL Best Effort mode; during each iteration Flent generated the TCP flows for 180 seconds. We also run one iteration of Flent for 300 seconds in the RRUL Ping mode.

B. Experiment Results on SWIFT Overheads

In Figure 7, we present the results of our experiments conducted to quantify the overheads incurred by our two techniques. In the Stock configuration, the packets between two clients associated with an AP by-pass the kernel network stack of the AP because they are forwarded directly by the radio interface driver. Our Intelligent AP technique causes these packets to be redirected to the *OVS* running on the AP, while our Thin AP technique causes these packets to be redirected to an external SDN switch. We quantify the impact of these redirections using the mean TCP goodput and RTT observed in the different scenarios discussed above.

In Figure 7(a) we present the RTTs observed during the *RRUL Ping* test. The mean TCP goodput and the mean RTT observed during the *RRUL (default)* test are presented in Figure 7(b) and Figure 7(c) respectively; the corresponding observations of *RRUL Best Effort* test are presented in Figure 7(d) and Figure 7(e) respectively. In each figure, `S`, `I`, and `T` denote the Stock, Intelligent AP, and Thin AP configurations respectively; cisco, ng, and tp denote the Cisco, Netgear, and TP-Link AP respectively. The numbers above the X-axis represent the percentage change in the value over the corresponding Stock configuration. For instance, in Scenario B when the Netgear and TP-Link APs are configured as Intelligent APs for the RRUL BE test (ng-tp,I-I), we observe a 1.2% decrease in the mean TCP goodput in Figure 7(d) and a 226% increase in mean RTT in Figure 7(e) compared to the mean goodput and mean RTT observed when the APs were used in their Stock configuration (ng-tp,S-S).

Scenario A. In this scenario, two clients associated with the same AP communicate with each other. The Thin AP configuration is expected to incur a larger increase in RTT compared to the Intelligent AP configuration because the packets have to traverse an external switch. This increase is clearly visible in Figure 7(a), Figure 7(c), and Figure 7(e); the increase in RTTs for the Intelligent AP configuration are smaller than the increase in the RTTs for the Thin AP configuration. Note that the amount of increase in RTT is dependent on the network latency between the AP and the external SDN switch; for instance, the latency between our external switch and our APs was under 1 ms. Furthermore, for the Cisco and Netgear APs, the impact of this redirection on the RTT is small compared to the increased queuing delays faced when the network is loaded; the increase in RTT for Cisco APs in the Thin AP configuration reduces from 55% under no load (see Figure 7(a)) to 0.2% under load from Best-Effort traffic (see Figure 7(e)). In RRUL Best Effort Test, the mean round-trip for the Netgear AP configured as a Thin AP is 5.7% lower while the TCP goodput is 1.4% higher than the corresponding RTT and goodput in the Stock configuration; we make similar observations in the RRUL Best Effort test when the Netgear AP is configured as an Intelligent AP. While the increases are mostly marginal, we believe that these are due to more optimal efficient processing of queues by *OVS*. In contrast, we observe an opposite behavior when using the TP-Link in the Intelligent AP mode. These changes in performance point to the hardware of the APs; the TP-Link has the weakest hardware in terms of CPU capacity, and this is evident from the high RTTs observed in the Stock configuration in Figure 7(a).

Scenario B and Scenario C. In Scenario B, the two laptops are associated with different APs. While in Scenario C, one laptop is connected to an AP and the other is in the wired network. With the help of Figure 6 one can see that for Scenario B the path traversed by the packets in the Stock-Stock configuration and Thin-Thin configuration are identical; similarly, the Stock-Wired configuration and Thin-Wired configuration are identical for Scenario C. The overheads incurred between these configurations are therefore marginal.

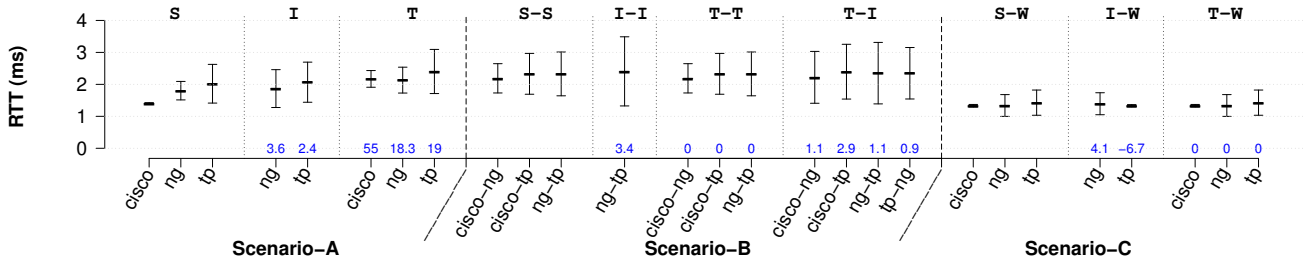
We observe a small difference between the TCP goodput in the Int-Int configuration and the goodput in the Stock-Stock configuration for Scenario B in Figure 7(b) and Figure 7(d). However, in Figure 7(c) and Figure 7(e), we can see that ICMP messages sent by Flent during the RRUL test and RRUL BE test incur a significantly higher RTT. Furthermore, for the Best Effort test, while the TCP goodput decreases by only 1.2% the RTT of the ICMP messages increases by 226%. Clearly, the Thin-Thin configuration performs better than the Int-Int configuration; *this highlights the benefits of offloading the flow management from commodity APs to an external switch*.

For Scenario C, the difference in the TCP goodput and RTT between the Intelligent-Wired configuration and the Stock-Wired configuration is marginal. In Scenario C, we observe a higher RTT compared to Scenario B and Scenario A because of the 1 Gbps wired link; the TCP flows from the faster wired network ensure that the queues at the AP are saturated.

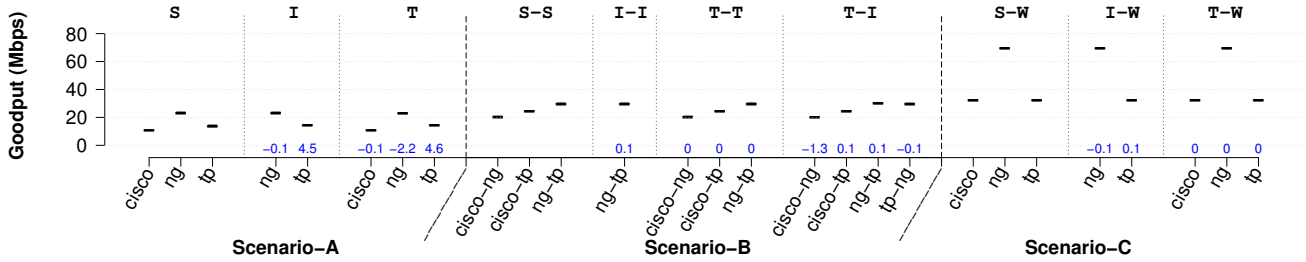
Across all scenarios, we observe that values for the TCP goodput and RTT in the RRUL test are significantly different from those in the RRUL Best Effort test. In the RRUL test, the ICMP messages used for the RTT measurements have the least priority and therefore have higher queuing delays. In the RRUL Best Effort test, all packets have the same priority; we therefore observe smaller RTTs compared to the RRUL test.

C. Experiment for Testing Scalability of SWIFT

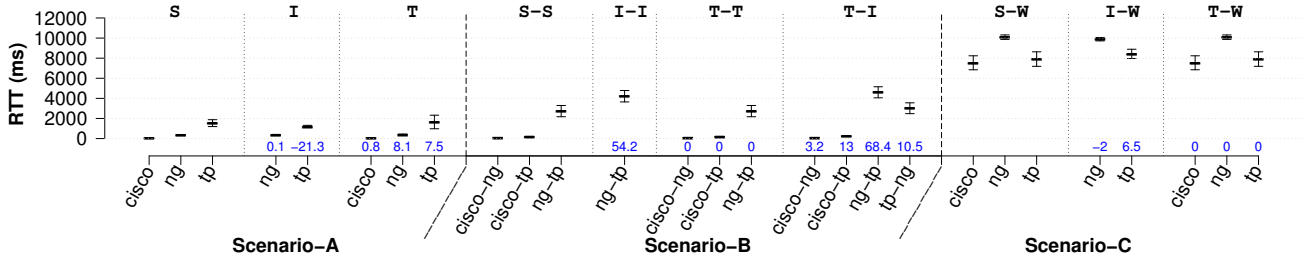
We also performed an experiment to showcase the scalability of our architecture. As discussed in §II, the previous



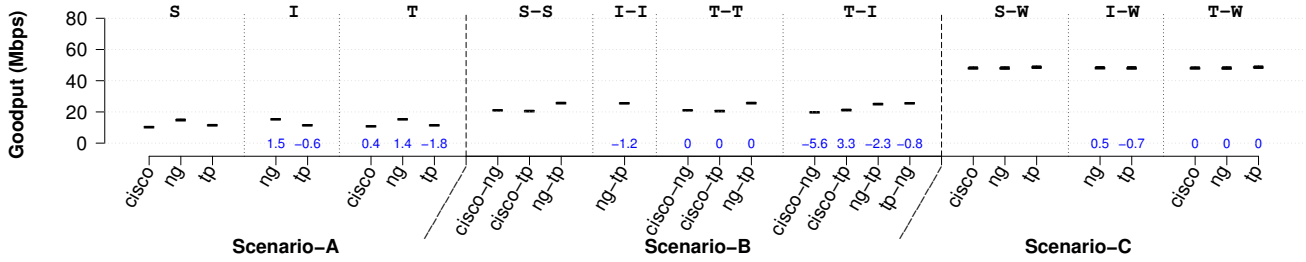
(a) **Mean round-trip times during RRUL Ping test.** *The error bars represent the standard deviation of the observed round-trip times. The numbers above the X-axis present the percentage change in the measured value (mean round-trip time) over the value observed in the corresponding stock configuration.*



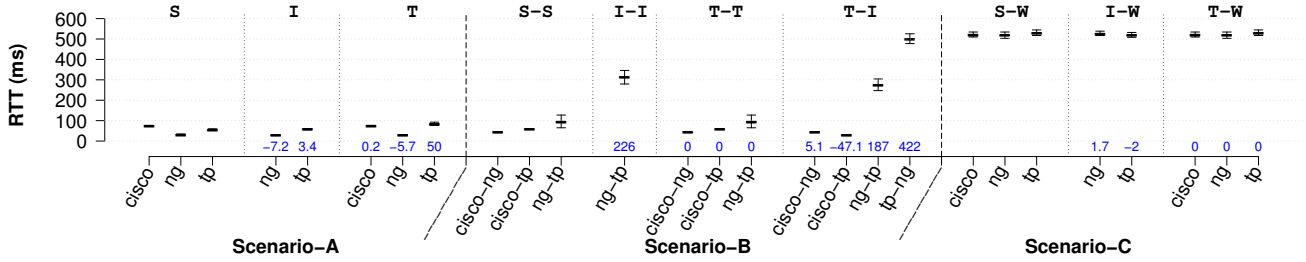
(b) **Mean goodput during the default RRUL test.** *The error bars represent 99% Confidence Intervals for the mean TCP goodput.*



(c) **Mean round-trip time during the default RRUL test.** *The error bars represent 99% Confidence Intervals for the mean round-trip time.*



(d) **Mean goodput during the RRUL BE test.** *The error bars represent 99% Confidence Intervals for the mean TCP goodput.*



(e) **Mean round-trip time during RRUL BE test.** *The error bars represent 99% Confidence Intervals for the mean round-trip time. Note the lower round-trip times compared to the default RRUL test. The ICMP messages have the same priority as the TCP flows in this test.*

Fig. 7. **Experiments Results**. *In each figure,* `S`*,* `I`*, and* `T` *denote the Stock, Intelligent AP, and Thin AP configurations respectively; cisco, ng, and tp denote the Cisco, Netgear, and TP-Link AP respectively. For instance,* `T-I` *with cisco-ng represents a scenario where the first client was associated with a Cisco AP configured as a Thin AP and the second client was associated with a Netgear AP configured as an Intelligent AP. Similarly,* `W` *represents a scenario when one of the clients was in the wired network. For instance,* `I-W` *with ng represents a scenario where one client was associated with the Netgear AP configured as an Intelligent AP and the second client was in the wired network. The numbers above the X-axis represent the percentage change in the value over the corresponding Stock configuration. For instance, in Scenario B when the Netgear and TP-Link APs are configured as Intelligent APs for the RRUL BE test (ng-tp I-I), we observe a 1.2% decrease in the mean TCP goodput in figure (d) and a 226% increase in mean round-trip time in figure (e) compared to the mean goodput and mean round-trip time observed when the APs were used in the Stock configuration (ng-tp S-S). We observe that our two approaches incur negligible overheads in terms of goodput, however their impact on the round-trip time varied with the tests.*

approaches are limited to a single client per VAP if all traffic flows need to be managed; however, the maximum number of VAPs is limited by the hardware and drivers of the APs. We show that our system can scale beyond the maximum number of available SSIDs on the Cisco AP hardware, *i.e.* beyond 16.

In the experiment, we used the same testbed as above with a single SSID. However, we used 20 different devices, including laptops, mobile phones, and a Chromecast. We associated the devices to the Cisco AP operating as Thin AP, and through the SWIFT controller we pushed isolation rules to the AP to isolate several of the devices from other devices.

As expected, the SWIFT controller was able to control all the traffic flows traversing the Thin AP. The isolated devices still retained the Internet access, but could not communicate with other devices, *i.e.* full control of the flows was achieved.

D. Evaluation Summary

The goal of our evaluation was to quantify the impact of our two techniques, and address the scalability of our architecture. The overhead results presented in this section show that our two techniques can be used on commodity APs. Furthermore, the differences in performance between Stock, Intelligent, and Thin configurations are negligible, and the performance largely depends on an AP's hardware capabilities. Specifically, the performance of the Intelligent AP technique depends heavily on an AP's hardware capabilities. In contrast, the Thin AP technique has fewer demands from the AP's hardware than the Intelligent AP, making it more suitable for APs with less powerful hardware. Our results also highlight the benefits of offloading the flow management from commodity APs to an external switch. *This implies that switches procured for SDN research can be used in Wi-Fi testbeds having commodity APs.*

The scalability experiment shows that our architecture can support networks with a large number of clients with only a single SSID, *i.e.* we can limit the overheads caused by a large number of SSIDs, and push beyond the hardware limitations of the APs which limit the other approaches.

Note that our goal was not to evaluate the performance of *OVS*; this has been done by Pfaff *et al.* [3]. We are also not quantifying the overheads of SDN policies because the observed latencies and goodputs can be policy specific and some policies can flood the switch with rules [26].

V. Conclusions

In this paper, we presented SWIFT, an architecture for bringing SDN-based traffic flow management to Wi-Fi networks built using commodity enterprise and consumer devices. SWIFT combines widely available technologies—Client Isolation and SDN switches such as *OVS*—and can therefore be used in almost all existing networks. The techniques used by SWIFT require only a small number of tasks, which are straightforward to implement in any SDN controller. We strongly believe that our techniques provide a deployable way to bringing SDN-based traffic management to Wi-Fi networks, and significantly lower the barrier-to-entry for conducting SDN research on Wi-Fi networks. For instance, our techniques can be used to bring new features such as AP-specific access control to enterprise networks, or conduct research on evaluating the benefits of SDN-based flow management in IoT networks which use Wi-Fi.

The instructions for deploying SWIFT are available at [27].

Acknowledgment

This work is supported by the Nokia Center for Advanced Research (NCAR), the Tekes PraNA project, and Tekes Take-5 project.

References

[1] "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," http://w1.fi/hostapd/.
[2] "OpenWrt," https://openwrt.org/.
[3] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *In Proc. of USENIX NSDI*, 2015.
[4] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "OpenRoads: Empowering Research in Mobile Networks," *ACM SIGCOMM CCR.*, pp. 125–126, 2010.
[5] M. Yan, C. J. Casey, P. Shome, A. Sprintson, and A. Sutton, "Ætherflow: Principled wireless support in SDN," *CoRR*, vol. abs/1509.04745, 2015.
[6] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, and N. McKeown, "BeHop: A Testbed for Dense WiFi Networks," in *Proc. of ACM WiNTECH '14*, 2014, pp. 1–8.
[7] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic Control over Home and Enterprise WiFi," in *Proceedings of SOSR '15*. New York, NY, USA: ACM, 2015.
[8] P. Calhoun, M. Montemurro, and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," *Internet RFCs*, vol. RFC 5415.
[9] ——, "Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Binding for IEEE 802.11," *Internet RFCs*, vol. RFC 5416.
[10] "Cisco Wireless LAN Controller," www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html.
[11] "Aruba Networks Mobility Controller," www.arubanetworks.com/products/networking/controllers.
[12] "Ubiquiti Networks UniFi," www.ubnt.com/enterprise.
[13] "Cisco Aironet 3700 Deployment Guide," http://www.cisco.com/c/en/us/td/docs/wireless/technology/apdeploy/8-0/Cisco_Aironet_3700AP.html.
[14] "SSID Overheads," http://www.revolutionwifi.net/revolutionwifi/2013/10/ssid-overhead-how-many-wi-fi-ssids-are.html.
[15] "Total wi-fi device shipments to surpass ten billion this month," http://www.wi-fi.org/news-events/newsroom/total-wi-fi-device-shipments-to-surpass-ten-billion-this-month, January 2015.
[16] "CPqD OpenFlow 1.3 Software Switch," http://cpqd.github.io/ofsoftswitch13/.
[17] "Linux Wireless," wireless.wiki.kernel.org.
[18] "Linksys WRT54G Series," https://en.wikipedia.org/wiki/Linksys_WRT54G_series.
[19] "The OpenDaylight Platform," https://www.opendaylight.org/.
[20] "Ryu SDN Framework," https://osrg.github.io/ryu/.
[21] D. McPherson and B. Dykes, "VLAN Aggregation for Efficient IP Address Allocation," *Internet RFCs*, vol. RFC 3069.
[22] S. HomChaudhuri and M. Foschiano, "Cisco Systems' Private VLANs: Scalable Security in a Multi-Client Environment," *Internet RFCs*, vol. RFC 5517.
[23] T. Høiland-Jørgensen, "Flent: The flexible network tester."
[24] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Communications*, pp. 57–65, January 2012.
[25] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2002, p. 213.
[26] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*, Oct 2010, pp. 408–415.
[27] "SWIFT," https://version.helsinki.fi/swift/.

Multipath IP Routing on End Devices: Motivation, Design, and Performance

Liyang Sun*, Guibin Tian*, Guanyu Zhu*, Yong Liu*, Hang Shi†, and David Dai†
*Electrical & Computer Engineering, New York University, Brooklyn, NY, 11201, USA
† Huawei Technology, Santa Clara, CA, 95050, USA

***Abstract*—Most end devices are now equipped with multiple network interfaces. Applications can exploit all available interfaces and benefit from multipath transmission. Recently Multipath TCP (MPTCP) was proposed to implement multipath transmission at the transport layer and has attracted lots of attention from academia and industry. However, MPTCP only supports TCP-based applications and its multipath routing flexibility is limited. In this paper, we investigate the possibility of orchestrating multipath transmission from the network layer of end devices, and develop a Multipath IP (MPIP) design consisting of signaling, session and path management, multipath routing, and NAT traversal. We implement MPIP in Linux and Android kernels. Through controlled lab experiments and Internet experiments, we demonstrate that MPIP can effectively achieve multipath gains at the network layer. It not only supports the legacy TCP and UDP protocols, but also works seamlessly with MPTCP. By facilitating user-defined customized routing, MPIP can route traffic from competing applications in a coordinated fashion to maximize the aggregate user Quality-of-Experience.**

I. Introduction

Contemporary end devices are normally equipped with multiple network interfaces, ranging from datacenter blade servers to user laptops and handheld smart devices. Exploiting all available interfaces, applications can adopt multipath transmissions to achieve higher and smoother aggregate throughput, resilience to traffic variations and failures on individual paths, and seamless transition between different networks. While each application can implement its own multipath transmission at the application layer, it is more desirable to provide multipath transmission services from the lower network protocol stack so that all applications can benefit. Recently, Multipath TCP (MPTCP) has been proposed and attracted lots of attention from academia and industry [1], [2], [3], [4], [5]. MPTCP allows all TCP-based applications enjoy the multipath gain in a *transparent* fashion. However, UDP-based applications cannot benefit from multipath transmissions.

In this paper, we share our experience of orchestrating multipath transmission from the network layer on end devices, and present a complete design of Multipath IP Transmission (MPIP). There are several advantages of implementing multipath transmission at the network layer:

Broader Coverage. MPIP can transmit IP packets generated by any TCP or UDP based application. Being transparent to the upper layers, MPIP can benefit all user applications without changing the application and transport layer protocols.

ISBN 978-3-903176-08-9

Better View and Coordination. The network layer can directly measure network status and promptly capture various dynamic events, such as interface and network changes. Since all application traffic go through the network layer, MPIP can adjust the transmission strategies for all applications in a coordinated fashion to maximally satisfy the diverse application and user needs.

More Flexible Routing. With MPTCP, traffic allocated to a path is determined by the rate achieved by the TCP subflow on that path, i.e., routing is simply determined by congestion control along multiple paths. This is too rigid and limited for applications with different throughput and delay requirements, and users with different resource and economic constraints. MPIP instead can implement any customized multipath routing.

Lower Complexity. MPIP can eliminate redundant network probings and routing adjustments attempted by individual applications and sessions. From the implementation point of view, similar to MPTCP, MPIP only requires changes on end devices. MPTCP has to work with the complexity resulted from the stateful TCP implementation. The legacy IP protocol is stateless and its implementation is much simpler than the legacy TCP. This leaves more design space for MPIP.

Meanwhile, MPIP also faces additional challenges. First of all, due to the stateless nature of IP, there is no existing session and path management mechanisms at network layer. Secondly, to work with multiple paths, MPIP constantly needs feedbacks about the availability and performance of each path. However, the legacy IP does not provide end-to-end feedbacks. Thirdly, various middle-boxes, e.g., NAT routers, are *by-no-means transparent*. They change and verify IP and TCP headers, and drop packets which they believe are "unorthodox" according to the legacy TCP/IP protocol. Multipath transmission unavoidably leads to out-of-order packet delivery. This will cause problem for running legacy TCP over MPIP. Finally, MPIP design and implementation should minimize the overhead and complexity added to the network layer. We address those challenges in our MPIP design and implementation. The contribution of our work is three-fold:

1) We develop a complete design to implement multipath transmission at the network layer, consisting of signaling, session and path management, multipath IP source routing, and NAT traversal. Our MPIP design not only can be used by the legacy TCP and UDP protocols, but also works seamlessly with MPTCP.

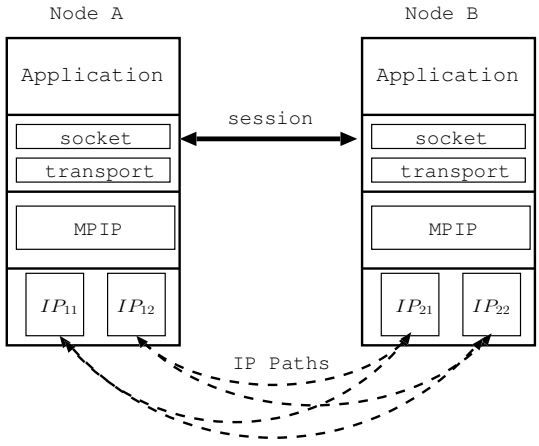


Fig. 1. Example of MPIP Transmission

2) MPIP supports diverse multipath routing strategies. For *all-paths* mode, we design a delay-based routing algorithm for MPIP to balance the loads of available paths. We also develop a user-defined multipath routing framework, through which customized routing strategies, such as *selected-paths* and *single-path*, can be realized by MPIP to satisfy diverse application/user needs.
3) We implement MPIP in Linux and Android kernels. We evaluate its performance using controlled lab experiments and Internet experiments. We demonstrate that MPIP can transparently achieve various multipath gains at the network layer. It works seamlessly with legacy transport layer protocols and popular applications. It can significantly improve user Quality-of-Experience (QoE) using easily configurable multipath routing strategies.

The rest of the paper is organized as follows. The semantics of MPIP is presented in Section II. The complete MPIP design is developed in Section III. Special issues related to TCP are addressed in Section IV. In Section V, we report the experimental results. Related work is summarized in Section VI. The paper is concluded in Section VII.

II. Semantics

MPIP works at the network layer on end devices. The basic building blocks are: *Node, Session, and Path.*

- *Node* refers to an end device with potentially multiple network interfaces, each of which gets assigned with a private or public IP address. MPIP also works with nodes with single network interface.
- *Session* is a transport layer flow between two nodes served by MPIP. A session is established at the transport layer, using the legacy TCP or UDP protocol, or even the new MPTCP protocol.
- *Path* is an end-to-end IP route available for a session. For each session, MPIP can use any interface on one node to transmit packets to any interface on the other node. If the two nodes have m and n interfaces respectively, the number of possible paths is mn.

With the legacy IP, each session is associated with only one IP (interface) and one port number on each node. The routing decision is based on destination IP address. MPIP employs customized *session-based* routing, and transmits packets of each session using any combination of the available paths. For the example in Figure 1, node A and node B are MPIP-enabled. They use the legacy application layer and transport layer. Each node has two interfaces (and the associated IP addresses). There are four end-to-end IP paths, as illustrated in Figure 1. When an application on node A opens a TCP/UDP connection to node B, MPIP will treat this connection as a new session. For each packet going from A to B, MPIP will choose one of the four available paths to send it out. To do that, MPIP will change the source and destination IP addresses as well as the port numbers of the packet so that it can be forwarded to the corresponding interface of the chosen path on node B. When node B receives the packet, it will first check which session it belongs to, then modify the IP address and port number back to the original values of the session. Finally, the packet will be passed to the corresponding TCP/UDP socket. The whole process is transparent to TCP/UDP session. If MPIP can simultaneously utilize the four paths by dispatching different packets to different paths, TCP /UDP throughput can be improved. Also the session can work normally as long as one path is available. Consequently, a TCP/UDP session will not be interrupted even if the default interfaces assigned to the session by the OS are disconnected. This makes handovers between different networks seamless and transparent to the transport and application layers. In general, MPIP routes packets from one session using several modes: 1) *all-paths mode:* packets are dispatched concurrently to all the available paths. Each packet will be transmitted along one of the paths. *MPIP Routing* determines the traffic splitting ratios among paths; 2) *selected-paths mode:* packets are routed on a subset of paths that meet the requirements of the application. Selected-paths mode avoids the inclusion of bad paths that will drag down the application performance. Path selection is application-specific and can be adapted by MPIP based on both application and network dynamics; 3) *single-path mode:* at any time, packets are only routed over one selected path, which can change during the course of the session. MPIP will handle seamless handover between paths, without interrupting the session. Single-path mode eliminates path quality disparity, such as out-of-order packet delivery, by sacrificing the throughput gain; 4) *protected-path mode:* a mission-critical packet is simultaneously transmitted on multiple paths. The receiver will pass the first arrived copy to the upper layer and discard the subsequent redundant copies. It sacrifices bandwidth for resilience.

III. MPIP Design

A. Workflow of Sending/Receiving Packets

Before diving into the design details, we present the MPIP workflow in Figure 2. When an outbound packet arrives at network layer from transportation layer, given the destination IP address and port number in header, MPIP checks whether

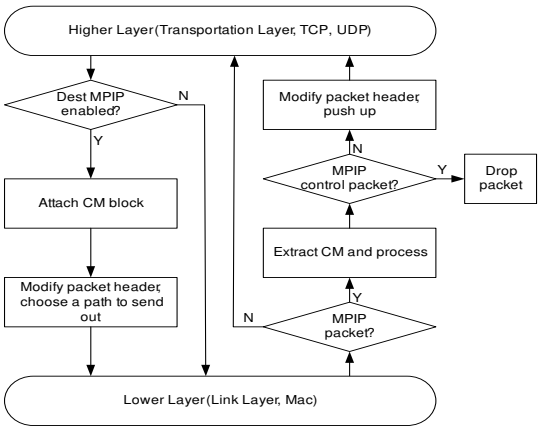


Fig. 2. MPIP Work Flow of Sending and Receiving Packets.

the destination node is MPIP enabled. If not, the packet will be processed by the regular IP stack and sent to the data link layer. If the destination is MPIP-enabled, MPIP will append a MPIP control message (CM) block to the end of the packet, change the IP and port addresses in packet header so that it will be sent to a chosen IP path. When receiving an inbound packet, MPIP processes the CM block to find the transport layer socket that the packet belongs to. Then MPIP reverts the IP and port addresses in packet header to the original values before pushing the packet to the transport layer. The major MPIP design components are: *Signaling Channel*, *Handshake*, *Session Management*, *Path Management*, *MPIP Routing*, and *NAT Traversal*.

B. Signaling Channel

TABLE I
CONTROL MESSAGE BLOCK

Source Node ID	*Session ID*	*Local IP Address List*	*CM Flags*
Path ID	*Feedback Path ID*	*Packet Timestamp*	*Path Delay*

MPIP needs realtime information about the availability and performance of end-to-end paths. Due to its connectionless design, legacy IP protocol doesn't have its built-in end-to-end feedback channel. We need a signaling channel for MPIP. Instead of transmitting extra signaling packets, we piggyback MPIP control information to each MPIP packet. For each packet sent out by MPIP, we add an additional control message (CM) data block at the end of user data. The size of the CM block is 25 bytes, a small overhead for typical data packets of $1000+$ bytes. Considering the throughput gain and robustness brought by MPIP, the overhead of CM block is well acceptable. Packet size may exceed the link MTU after attaching the CM block. We force the transport layer to reduce the size of each segment, e.g. decreasing the MSS value for TCP connection, to make sure the CM block fits within the MTU limit. The information contained in a CM block of a packet is shown in Table I.

Source Node ID is a globally unique ID of the sending node of this packet. Since each node has multiple interfaces, and their IP addresses may change over time, to have a semi-static node ID, we use the MAC address of a NIC (preferable more static ones) on the node to be its ID.

Local IP Address List carries all local IP addresses on the sending node. This list will be used to construct MPIP paths.

CM Flags encodes the MPIP functionality of the packet. With different values of *CM Flags*, different actions will be operated when the packet is received.

Other fields will be explained in the following sections.

C. Handshake and Session Management

As an extension of IP, MPIP needs to be backward compatible. To take advantage of MPIP, both end nodes of a session need to be MPIP-enabled. Locally, every MPIP-enabled node maintains a table to record the availability of MPIP on remote nodes. A node can query the MPIP availability of a remote node by sending out a MPIP packet with $Flags\_Enable$ in CM. If the remote node is MPIP-enabled, it will send back confirmation. Both nodes will update their MPIP availability table accordingly. Please refer to our technical report [6] for the detailed handshake process, After the MPIP handshake, a node can start to learn the interfaces available on each MPIP-enabled remote node. Each node maintains a node ID to IP address and port number mapping table. Every time a MPIP packet is received, the receiver extracts the sender's node ID from the packet's CM block, and IP address and port number from the packet header. The three tuple is then written into the mapping table.

MPIP conducts session-based routing. Session management takes care of the addition and removal of TCP and UDP sessions. At the transport layer, each session is identified by the traditional 5-tuple: source and destination IP addresses and port numbers, and protocol type. Since MPIP can transmit a packet of a session using source and destination IP address/port numbers different from the session's original ones, we can no longer use IP addresses/port numbers to associate a MPIP packet with a transport layer session. Instead, we will use session ID and node ID carried in the CM block to identify the session of a MPIP packet. We need a table to correlate the two different session mapping schemes employed by MPIP and the legacy transport layer. This is achieved through the session information table, as in Table II. The table maintains one entry for each session to each remote node. For each entry, the socket information, namely IP addresses and port numbers, are the original ones from the transport layer.

After the MPIP availability handshake has been successfully completed, when sending out a packet, the sender checks Table II to see whether a proper session entry has been generated. If not, MPIP generates a new session ID and adds a new entry to Table II. After this, all packets belong to the session will carry the session's ID in its CM block. On the receiver end, whenever a MPIP packet is received, the receiver extracts the source node ID and session ID from its CM block. If there is no entry found in its session information table, it

TABLE II
SESSION INFORMATION TABLE

Dest. Node ID	Session ID	Source IP	Source Port	Destination IP	Destination Port	Protocol Type	Next Sequence No	Update Time
ID_1	SID_1	SIP_1	$SPORT_1$	DIP_1	$DPORT_1$	TCP	S_1	T_1
ID_1	SID_2	SIP_1	$SPORT_2$	DIP_1	$DPORT_2$	UDP	0	T_2
ID_2	SID_1	SIP_2	$SPORT_3$	DIP_2	$DPORT_3$	TCP	S_2	T_3
ID_2	SID_2	SIP_2	$SPORT_4$	DIP_2	$DPORT_4$	UDP	0	T_4

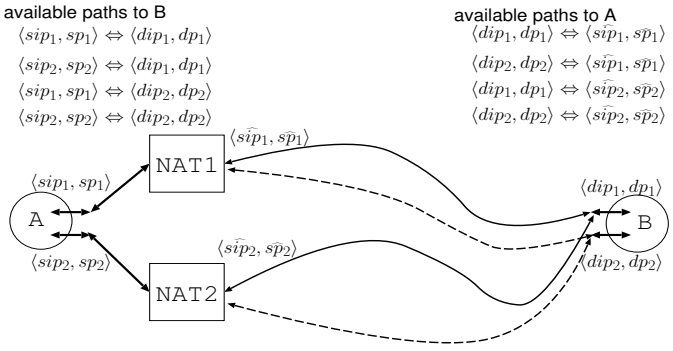


Fig. 3. MPIP Path Establishment with NAT

will generate a new entry and populate it with the source node ID, session ID, and socket information carried in the packet header, with swapped source and destination IP/port addresses. This will make sure that both sides of the same session use the same session ID. Removal of a session is done by expiration based on the session's *Update Time* in Table II. The column *Next Sequence No* is used for TCP out-of-order process which will be explained in Section IV-B.

D. Path Management

After a session is registered with MPIP, the next step is to explore all the available paths for the session. One simple solution is to have each node send their local IP addresses to the other end using the *Local Address List* in CM block. Then any pair of IP addresses on the two ends can be used as a path for MPIP transmission. However, this only works if all interfaces on both ends have public IP addresses. If one node is behind a NAT, its local IP addresses cannot be used directly to establish IP paths. To solve this problem, we have to identify paths using a combination of IP address and port number on both ends. Consequently, the path management has to be done for each session individually.

1) Establishment: MPIP maintains a path information table on each node, as in Table III, to record the available paths for each session. Each entry contains the ID of the remote node and the session ID. Each path is allocated with a path ID, which is unique on the local node. The source and destination IP and port addresses are the addresses carried in the header of MPIP packet, NOT necessarily the same as those allocated to the session at the transport layer.

Given m and n interfaces at each end node, there are totally mn possible paths. After the MPIP handshake, each node tries to send out packets from each of its local interfaces to each of the known interface on the remote node. If a packet with a certain combination of source and destination IP/port addresses can get through, the node will add the path to path information table. Let's explain the process through the example in Figure 3. Node A initiates a session with node B. The IP and port addresses allocated to the session at the transport layer are $\langle sip_1, sp_1\rangle$ and $\langle dip_1, dp_1\rangle$ on A and B respectively. Without loss of generality, let's assume the session can be established correctly with legacy IP. Then on both ends, MPIP records the new session, and adds the default path between $\langle sip_1, sp_1\rangle$ and $\langle dip_1, dp_1\rangle$ for the session in Table III. Since A knows B is MPIP-enabled, it also tries to send the same packet from its other local interface with IP address sip_2 by changing its source addresses to $\langle sip_2, sp_2\rangle$. When B receives the packet, possibly due to NAT, the source IP and port addresses in the packet might be different from $\langle sip_2, sp_2\rangle$, say $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$. Then B examines the *Source Node ID* and *Session ID* in the packet's CM block, it knows this is a MPIP transmission for the same session but from a different interface. B adds a new path with destination address of $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$ in its path information table. Now B will also send back packets to A's second interface, using destination addresses $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$. When A receives the packet, it confirms the connectivity of its local path between $\langle sip_2, sp_2\rangle$ and $\langle dip_1, dp_1\rangle$, and adds it to its path information table. Similarly, if B has another interface with public address dip_2, A will obtain the new address from the *Local Address List* in the CM block of packets from B to A. Then A can establish more IP paths to this new address using a similar process.

2) Monitoring: To facilitate path selection, MPIP continuously monitors the performance of active paths. Given that packet losses in the current Internet are rare, we mainly focus on path delay in our current design. Due to asymmetric routing and unequal congestion levels along two directions of the same path, instead of measuring the round-trip delay of a path, we measure the one-way path delay to infer the path quality on each direction. When node A sends out a packet, it chooses a path from Table III and sets *Packet Timestamp* with its local system time T_1. After node B receives this packet, it calculates the one-way delay for the path as $T_2 - T_1$, where T_2 is B's local time when receiving the packet. In practice, the absolute value of path delay calculated here isn't the real delay value because of the clocks on node A and B are not synchronized. But our path selection algorithms depend on the relative ordering of path delays, instead of their absolute values. Clock difference between nodes has little impact. B then sends back the path delay information in the CM block

TABLE III
PATH INFORMATION TABLE

Dest Node ID	Session ID	Path ID	Src IP	Src Port	Dest IP	Dest Port	Minimum Path Delay	Real-Time Path Delay	Real-Time Queuing Delay	Maximum Queuing Delay	Path Weight
ID	SID_1	PID_{11}	sip_1	sp_1	dip_1	dp_1	D_{min11}	D_{11}	Q_{11}	$Q_{max_{11}}$	W_{11}
ID	SID_1	PID_{12}	sip_2	sp_2	dip_1	dp_1	D_{min12}	D_{12}	Q_{12}	$Q_{max_{12}}$	W_{12}
ID	SID_2	PID_{21}	sip_1	sp_1	dip_2	dp_2	D_{min21}	D_{21}	Q_{21}	$Q_{max_{21}}$	W_{21}
ID	SID_2	PID_{22}	sip_2	sp_2	dip_2	dp_2	D_{min22}	D_{22}	Q_{22}	$Q_{max_{22}}$	W_{22}

of the next packet going back to A, which records the path delay value into the column *Real-Time Path Delay* in Table III. Path delay values are smoothed using a simple moving average algorithm. More details can be found in our technical report [6]

3) Dynamic Path Management: MPIP supports dynamic addition and removal of paths from Table III. When IP address change happens on one node, it sets $Flags\_IP\_Change$ in the CM block of its next outgoing packet. After receiving a packet with this flag, the receiver knows that IP address on the sender has changed, it removes all path entries related to the changed IP address in Table III. Meanwhile, the entry for this session in Table II remains unchanged. The path that sends out the IP change notification will be added back to the aforementioned tables as the only path of the session. Also, the sender does the same reset for this session. After all these resets, there is only one path left for this session, all the other available paths will be added back through the procedure in Section III-D1. Similarly, when a new interface becomes available, new IP paths from it can be added using the the mechanism in Section III-D1. Table III should be updated continuously on both sides. The updates are piggybacked on MPIP packets. For sessions with one-way traffic, such as some UDP sessions, a periodical heartbeat mechanism is introduced to keep Table III fresh. More details can be found in our technical report [6].

E. Multipath IP Source Routing

Given all paths available for a session, every time one node needs to send out a packet, it chooses the most suitable path from Table III. MPIP offers different routing strategies to satisfy the diverse needs of applications.

1) All-paths Mode: Many applications, e.g., web, file transfer, and video streaming, can benefit from high-throughput transmissions. MPIP can concurrently transmit packets along multiple paths to achieve higher throughput than the traditional single path routing. Since MPIP works under rate control schemes from transport and application layers, it will be redundant and possibly conflicting to implement fine-grained rate control for each MPIP path at the network layer. Instead, the main design goal of MPIP routing is to balance load among concurrent paths using end-to-end path delay feedback and probabilistic packet dispatching algorithm. As in Table III, we maintain a *Path Weight (W)* for each active path. Each packet will be dispatched to a path k with the probability $P(k)$, which is calculated as:

$$P(k) = \frac{W_k}{\sum_{i=1}^{N} W_i}. \quad (1)$$

We use realtime one-way path delay to dynamically update path weights. End-to-end path delay consists of propagation delay, transmission delay, processing and queueing delay. While propagation delay and transmission delay are mostly constant, processing and queue delay are time-varying and increase with congestion level. We maintain the minimum path delay to represent the constant portion of end-to-end path delay, and use the difference between real-time and minimum delay to infer the queuing delay, which reflects the congestion level along the path. We then adjust the weight of each path using the real-time queuing delay. When a new delay sample D is received, the other three delay metrics are updated:

1) *Minimum Path Delay:* $D_{min} = \min\{D_{min}, D\}$;
2) *Real-Time Queuing Delay:* $Q = D - D_{min}$;
3) *Maximum Queuing Delay:* $Q_{max} = \max\{Q_{max}, Q\}$.

We adjust the weights of all paths together based on their queueing delay variations as in Algorithm 1. N is the number

Algorithm 1 Path Weight Adjustment.

```
1: Q_avg = (Σ_{i=1}^{N} Q_i)/N; //average delay among all paths
2: if Q_i ≤ Q_avg then
3:    W_i = W_i + S; //increase weight for low delay path
4:    if W_i > 1000 then
5:       W_i = 1000; //upper bound for path weight
6:    end if
7: else
8:    W_i = W_i − S; //decrease weight for high delay path
9:    if W_i < 1 then
10:      W_i = 1; //lower bound for path weight
11:   end if
12: end if
13: return ;
```

of paths that belong to one session, Q_i and W_i are queuing delay and weight of path i, and S is the adjustment granularity. Initially, every path has the same path weight of $\frac{1000}{N}$. In each iteration, the path weight increases or decreases by S based on whether its queuing delay is higher or lower than the average delay. The maximum weight is 1000, and the minimum is 1. This way, we keep all live paths in consideration. Heavily congested paths will not be completely eliminated. Instead they will have the minimum weight, and their weights will be increased after congestion is relieved. Algorithm 1 is executed periodically, the length of each period is defined as a configurable system parameter T.

2) User-defined Multipath Routing: Not all applications take throughput as the first priority. To address the diverse needs of applications, we design MPIP to support user-defined routing schemes, including *selected-paths*, *single-path* and *protected-path*. Users can inform MPIP of their desired multi-path routing policies by configuring a routing table as illustrated in Table IV. Each line of the table is a customized

TABLE IV
USER-DEFINED MULTIPATH ROUTING TABLE

IP Address	Port Number	Protocol Type	Start Size	End Size	Routing Priority
$*$	22	TCP	0	200	R_f
192.168.1.2	5222	UDP	200	$*$	T_f
192.168.1.2	5221	UDP	0	500	R_f

routing rule for outgoing packets. Each rule matches a set of packets and the routing priority for the matched packets. Packet matching is done using destination IP address, port number, protocol, and the range of packet length. We currently define two types of routing priorities: *throughput-first* T_f, and *responsiveness first* R_f. Outgoing packets with T_f priority will be dispatched to available paths using the *all-paths* mode presented in Section III-E1. Outgoing packets with R_f priority will always be sent to path with the lowest delay using the *single-path* mode. For example, based on the first row of Table IV, for any TCP connection with destination port 22 (ssh session), if the packet length is smaller than 200 bytes, the packet will be forward to the lowest delay path. The second row defines that all UDP packets going to a remote host with packet size larger than 200 bytes should be forwarded using *all-paths* mode. The third row specifies that for a UDP packet going to the same remote host, but a different port number, if the packet size is less than 500, it will be forwarded to the lowest delay path instead. We will extend this basic framework to incorporate more flexible and more user-friendly packet matching rules and more diverse routing policies with finer granularity in our future work.

IV. TCP-RELATED ISSUES

By deviating from the default single-path transmission, MPIP also brings some new issues for the upper layer protocols, especially TCP, such as NAT checking and out-of-order packet delivery. It is also intriguing to explore the co-existence of MPIP with multi-path transmissions at upper layers, such as MPTCP. We now present solutions to TCP-related issues.

A. NAT Checking

Based on our experiments and other studies, e.g. [1], NAT devices are by no means transparent, and conduct all kinds of mapping, verification, and dropping to end-to-end sessions, especially TCP. One immediate obstacle introduced by NAT to MPIP is that many NAT devices drop a TCP packet if they don't have a record about the TCP connection that the packet belongs to. In MPIP, if we transmit TCP packets on a path different from the original one through which the TCP connection is established, NAT devices along the path are not aware of the connection and will drop these packets before they arrive at the destination. We provide two solutions. One solution is to construct a fake TCP three-way hand-shake on the NAT's path before sending packets over. When a client receives the IP address list of the server, it sends out a SYN packet along each possible path to the server except the original one which was used to initiate the real TCP connection. After the fake three-way handshake is completed successfully, NAT routers along the path have a record about this fake TCP connection, will pass TCP packets assigned to the path. Another solution is UDP wrapper. During our experiments, most NAT devices don't verify socket information of UDP packets. We make use of this feature and wrap a TCP packet inside a UDP packet to pass NAT checking. Whenever MPIP chooses for a TCP packet a path different from its original path, it encapsulates the TCP packet into an UDP packet by adding a forged UDP header using the corresponding IP addresses and port numbers of the chosen path. At the receiver end, MPIP removes the UDP header and extract the original socket information from Table II to be filled into the TCP and IP headers.

B. Out-of-order Packet Processing

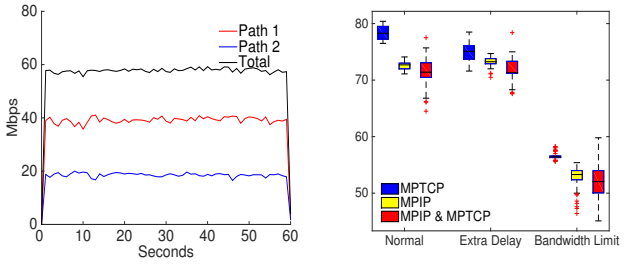
Packets sent over multiple interfaces/paths can arrive at the destination node out of order. When TCP works over MPIP, if the delay difference between multiple paths is significant, we can expect a lot of out-of-order packets. To resolve this problem, for each session in Table II, if it is TCP protocol, MPIP maintains the sequence number S of the next in-order packet of the session to be received. MPIP also maintains a separate re-sequencing buffer B for each active session to store out-of-order packets. Whenever a new packet is received, if the sequence number is larger than S, it will be stored in B; if the sequence number equals to S, MPIP pushes all consecutive packets in B to the transport layer and update S accordingly. To avoid blocking introduced by a lost packet, we limit the size of re-sequencing buffer. All the packets in the buffer will be pushed up once the buffer is full. In our prototype, we set the maximum buffer size to 100 packets.

C. MPTCP over MPIP

A MPTCP session employs multiple subflows, each of which is a legitimate TCP connection over a single IP path. When MPTCP runs over MPIP, each TCP subflow can now utilize multiple paths. For the example in Figure 1, a MPTCP session can have 4 subflows. MPIP will treat each subflow as an independent TCP session, and will create 4 paths for each subflow. As a result, there are totally 4 sessions and 16 paths managed by MPIP. When congestion accumulates on one path, MPIP will first notice the high queuing delay on that path, reduce the path weight and shift packets to less congested paths. The load balancing conducted by MPIP at the network layer makes the congestion variations along different paths less perceivable for MPTCP subflows so that MPTCP can make better use of subflows to achieve higher throughput. We will demonstrate this using MPTCP+MPIP experiments in Section V-A1.

V. Performance Evaluation

To evaluate the performance of the proposed design, we implement MPIP in Linux kernel 3.10.11 in Ubuntu system for IPv4. The main MPIP functions are implemented with more than $5,000$ lines of code. MPIP is also implemented into Android system 6.0.1 with kernel version 3.10.73. For all TCP experiments, we use CUBIC-TCP [7]. MPTCP version 0.92 is used in our evaluation. We use *Iperf/Iperf3* to generate traffic.



(a) Load Balancing (b) MPIP and MPTCP

Fig. 4. TCP over MPIP Performance

A. Controlled Lab Experiments

In our lab, we install the prototype on two desktop computers, which are connected directly to a router. Each desktop has two 100Mbps NICs, leading to 4 paths with aggregate capacity of 200Mbps. We use *tc (traffic control)* tool in Linux to control bandwidth and delay on each path.

1) TCP over MPIP: To test the effectiveness of MPIP load-balancing, we enable only two parallel paths between the two desktops so that they don't share any NIC to prevent traffic coupling. To make it more intuitive, we limit the bandwidth of path 1 to 40Mbps and path 2 to 20Mbps. From the throughput trend in Figure 4(a), both paths converged close to their capacities and remained stable for the whole experiment. We then compare path failure response time of TCP/MPIP and MPTCP/IP by disconnecting then reconnecting one path. MPTCP always suffers a $10 \sim 20$ seconds delay to re-establish the subflow. MPIP promptly detects the re-activated path at the network layer to ramp up the throughput.

As mentioned in Section IV-C, MPIP should be compatible with MPTCP. Three groups of experiments are conducted for different combinations of multipath transmission at transport and network layers, namely, MPTCP/IP, TCP/MPIP, and MPTCP/MPIP. For the first group (normal), two available paths with 40Mbps bandwidth each are configured; for the second group (extra delay), an extra 10ms delay is added to path 1; at last, bandwidth of path 1 is limited to 20Mbps. In Figure 4(b), the boxplots for throughputs of all combinations are plotted. MPTCP/IP throughput is stable and close to the capacity in all cases. TCP/MPIP and MPTCP/MPIP throughputs are little lower but still close to the capacity. Their throughput variances are also larger than MPTCP. The interaction between MPIP load balancing and upper layer congestion control needs further study and fine-tuning.

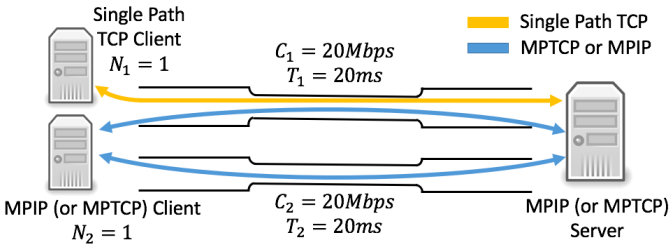
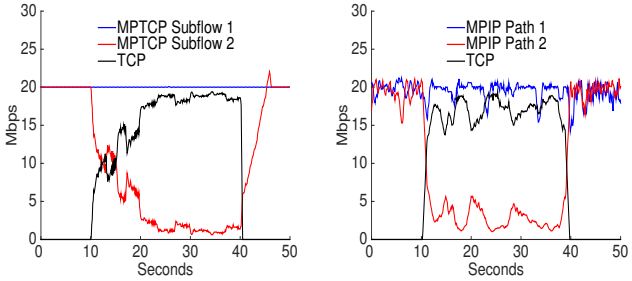


Fig. 5. MPTCP/MPIP Compete with Single Path TCP



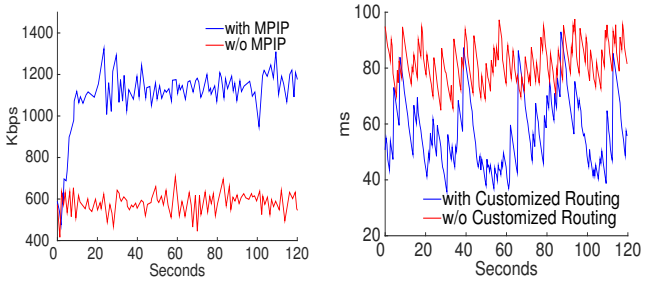
(a) MPTCP/IP and TCP/IP (b) TCP/MPIP and TCP/IP

Fig. 6. Fairness with Legacy TCP/IP

2) Fairness with Legacy TCP/IP: We next conduct experiments to study how TCP/MPIP co-exists with legacy TCP/IP sessions, and compare it with MPTCP. Consider a network containing three types of sessions, TCP/IP, MPTCP/IP, and TCP/MPIP, illustrated in Fig 5. Similar to the MPTCP fairness study in [8], two paths are set up with two bottleneck links of $20Mbps$. The upper path is shared by the TCP/IP session and MPIP (or MPTCP) session. The MPIP (MPTCP) session starts first. The TCP/IP session follows after ten seconds, and lasts for thirty seconds. Fig 6(a) illustrates how MPTCP with BALIA congestion control (CC) co-exists with TCP. MPTCP gradually reduces its traffic on the shared path to leave space for the single-path TCP, which eventually gets comparable throughput as MPTCP. When TCP session is done, it takes a while for MPTCP to reclaim the capacity on the shared path. Meanwhile, from Fig 6(b), MPIP reacts much faster than MPTCP to make space for single-path TCP, which obtains nearly all the available bandwidth of the shared link. After single-path TCP completes, MPIP also reclaims the available bandwidth faster than MPTCP. This demonstrates that MPIP's load balancing at the network layer can facilitate fair bandwidth sharing at the transport layer.

3) UDP over MPIP: To evaluate how UDP-based applications, such as Real Time Communications, can benefit from MPIP, we run WebRTC video chat over MPIP and collect application-level performance by capturing the statistics windows of WebRTC-internals embedded in Chrome, then extracting data from the captured windows using WebPlotDigitizer. We first configure two IP paths between two lab machines without bandwidth limit, and then run WebRTC video call between the two machines. To test the robustness of MPIP

against path failures, one path is disconnected in the middle of experiment. If WebRTC video chat is running over legacy IP, when the original path is disconnected, video freezes for few seconds before video flow migrates to the other path. This demonstrates that while WebRTC can recover from path failure at the application layer, its response is too sluggish and user QoE is significantly degraded by a few seconds freezing. With MPIP, video streams continuously without interruption. In addition, to demonstrate how WebRTC benefits from MPIP multipath throughput gain, we limit the bandwidth of each path to 1Mbps. Comparison presented in Figure 7(a) illustrates that with the help of MPIP, WebRTC video throughput improves from 600Kbps to 1200Kbps. We then introduce additional delays of 50ms and 80ms to the two paths respectively. MPIP then use *single-path* mode to route audio packets to the path with shorter delay, while video packets are routed using *all-paths* mode. Figure 7(b) shows clearly that audio delay is reduced by 30ms while the video quality is not affected.



(a) Video Rate under B.W. Limit (b) WebRTC Audio Delay

Fig. 7. WebRTC Performance over MPIP: (a) all-paths mode; (b) single-path mode for audio, all-paths mode for video.

B. Internet Experiments

Besides the controlled lab experiments, we also conduct experiments on the Internet to evaluate MPIP's compatibility with real applications and various middle boxes, e.g. NAT routers inside ISP and CSP networks.

1) Coordinated Routing between Applications: We study coordinated MPIP routing for Youtube video streaming and file downloading applications using the testbed in Figure 8. Since we cannot install MPIP on YouTube servers, we configure a MPIP proxy using Squid on Ubuntu. Three NICs are installed on the proxy server: one NIC is connected to Internet, and the other two are connected to a MPIP client through two paths in an emulated network. We setup 2Mbps bandwidth limit for each path and introduced 20ms extra delay to one path.

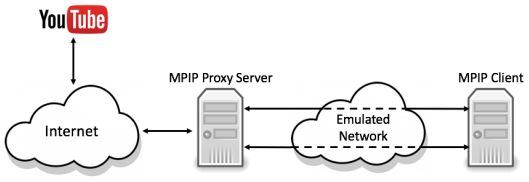
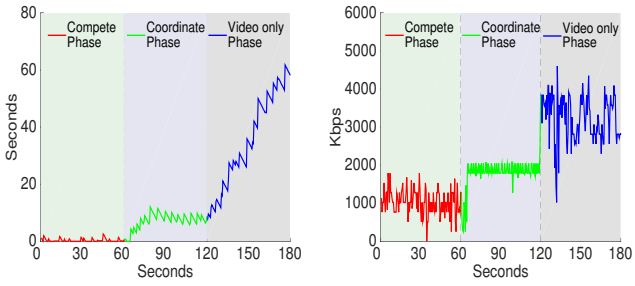


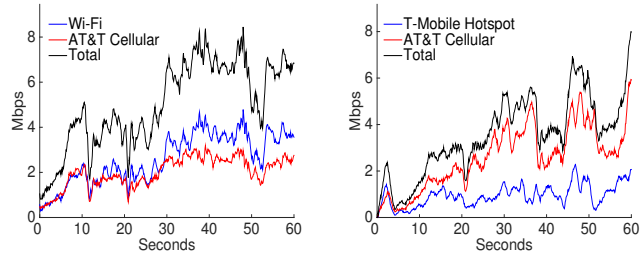
Fig. 8. MPIP works with YouTube through Proxy



(a) Video Buffer Health (b) Video Throughput

Fig. 9. Youtube 720p Video Streaming with Coordinated MPIP Routing

At the beginning, besides the YouTube video session, another file downloading session is added to transmit data from MPIP proxy server to client. Initially MPIP operates in the *all-paths* mode and establishes two paths for each session to acquire more bandwidth. Due to the path delay difference, out-of-order packet deliveries limit the TCP throughput for both sessions. Sixty seconds into the experiment, MPIP implements *coordinated routing*: both sessions are routed using the *single-path* mode, with Youtube session assigned to the path with shorter delay and the file downloading session assigned to the other path. In Figure 9, coordinated routing significantly improve the performance of the video session: video throughout increases by 400Kbps (from $1,500$Kbps to $1,900$Kbps), and buffer length accumulates to 10 seconds without freezing. Meanwhile, the average throughput of the downloading session drops from 2.51Mbps to 1.89Mbps. Since users are more sensitive to video quality than the file downloading throughput, the coordinated routing presumably improves the overall user experience. Sixty seconds later, we terminated the downloading session. From Figure 9(a) and 9(b), we observe that both the video throughput and preload buffer length increase significantly.



(a) WiFi and Cellular (b) Two Cellular Networks

Fig. 10. MPIP over Wireless

2) Android Experiments: We use a Nexus 5X phone located in California to test Android MPIP. The phone is equipped with one cellular interface and one WiFi interface. We use it to download data from a server located in New York City with one public IP address. We first connect the phone to a corporate ISP through WiFi and AT&T CSP through 4G cellular. Without MPIP, the phone can achieve average

bandwidth of 4.5Mbps through WiFi and 4.3Mbps through cellular respectively. The average RTTs of WiFi and cellular are 76.2ms and 155.9ms respectively. When MPIP is enabled, as illustrated in Figure 10, Android MPIP can concurrently transmit data on both paths going through different ISP/CSP and reach aggregate throughput of 7.5Mbps in the face of large delay disparity. Next we replace the corporate WiFi router with a hotspot hosted by another phone connected to T-Mobile cellular network. As all data through the hotspot are forwarded by another phone, the average RTT on the T-Mobile path increases dramatically to 349.2ms and the average bandwidth is only 1.52Mbps. Figure 10(b) demonstrates that even when one cellular path has bad performance, MPIP still manages to multiplex bandwidth from two CSPs to achieve higher aggregate throughput.

VI. Related Work

The growing popularity of multi-homed devices makes it possible to initiate multipath transmission from end devices. Back to 2001, Hsieh et al proposed pTCP[9] that effectively performs bandwidth aggregation on multi-homed mobile hosts. In [10], the authors investigated the potential benefits of coordinated congestion control for multipath data transfers. In [11], Dong et al implemented concurrent TCP(cTCP) in FreeBSD to improve throughput. Also, the Stream Control Transmission Protocol (SCTP)[12], [13] is an early protocol designed for multihoming to support failover and simultaneous transmission. In 2010, Barre et al published experimental results of using multiple paths simultaneously in TCP transmission [4], [1]. IETF RFC 6182 [14] for Multipath TCP was published in in 2011. In [2], Chen et al did a thorough measurement of MPTCP over wireless links. Different from those multipath protocols at the transport layer, MPIP is a transparent multipath solution at the network layer of end devices. As bandwidth of cellular network becomes comparable with the wired Internet, switching among WiFi and cellular becomes practical for mobile devices, e.g. [15], [16]. All these solutions require significant changes and coordination at multiple layers. In [17], a pure user-level solution, called msocket, was proposed for seamless handover between different mobile networks. Different from these previous work, MPIP realizes path selection and seamless handover by only changing the network layer. It has long been observed that routing for applications on the same device needs to be coordinated [18], [19]. MPIP serves as a light-weight framework to implement coordinated routing for multiple applications.

VII. Conclusions and Future Work

In this paper, we developed MPIP, a complete design of multipath transmission at the network layer of end devices. MPIP consists of signaling, session and path management, multipath routing, and NAT traversal. MPIP can be used by both TCP and UDP-based applications. It also works seamlessly with MPTCP, and supports user-defined routing strategies. We implemented MPIP in Linux and Android kernels. Through extensive lab and Internet experiments, we demonstrated that MPIP can transparently support flexible and coordinated routing for diverse applications to achieve multipath gains. MPIP is only our first attempt for implementing multipath transmission at the network layer. The signaling and feedback mechanisms can be further optimized to reduce its overhead and improve its robustness. The delay-based load balancing algorithm can be improved to better address path heterogeneity, especially for WiFi, LTE, and the emerging 5G Cellular links. We will extend the user-defined routing framework to support finer routing granularity and more flexible forwarding actions. We will also port MPIP to IPv6. Finally, we will further study the efficiency, fairness and stability of the vertical and horizontal interactions of MPIP with legacy TCP and IP protocols through analysis, simulations and prototype experiments.

References

[1] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp," in *NSDI*, 2012.

[2] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *IMC*, 2013.

[3] Y.-C. Chen, D. Towsley, E. M. Nahum, R. J. Gibbens, and Y.-s. Lim, "Characterizing 4g and 3g networks: Supporting mobility with multipath tcp," *School of Computer Science, University of Massachusetts Amherst, Tech. Rep*, vol. 22, 2012.

[4] S. Barre, C. Raiciu, O. Bonaventure, and M. Handley, "Experimenting with multipath tcp," in *SIGCOMM 2010 Demo*, September 2010.

[5] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath tcp," in *CoNEXT*, 2013.

[6] L. Sun, G. Tian, G. Zhu, Y. Liu, H. Shi, and D. Dai, "Multipath IP Routing on End Devices: Motivation, Design, and Performance," Tandon Engineering School, New York University, Tech. Rep., 2017, available at http://eeweb.poly.edu/faculty/yongliu/docs/MPIP_Tech.pdf.

[7] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, Jul. 2008.

[8] Q. Peng, A. Walid, J.-S. Hwang, and S. H. Low, "Multipath tcp algorithms: Theory, design and implementation," *IEEE/ACM Transactions on Networking*, 2016.

[9] H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," in *MobiCom*, 2002.

[10] P. Key, L. Massoulié, and D. Towsley, "Path selection and multipath congestion control," *Commun. ACM*, vol. 54, no. 1, Jan. 2011.

[11] Y. Dong, D. Wang, N. Pissinou, and J. Wang, "Multi-path load balancing in transport layer," in *Next Generation Internet Networks, 3rd EuroNGI Conference on*, May 2007.

[12] L. Ong, C. Corporation, and J. Yoakum, "An introduction to the stream control transmission protocol (sctp)," IETF RFC 3286, 2002.

[13] I. Joe and S. Yan, "Sctp throughput improvement with best load sharing based on multihoming," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, Aug 2009.

[14] A. Ford, C. Raiciu, M. Handley, S. Barre, U. C. D. Louvain, and J. Iyengar, "IETF RFC 6182: architectural guidelines for multipath tcp development," 2011.

[15] P. Nikander, T. Henderson, C. Vogt, and J. Akko, "End-host mobility and multi-homing with host identity protocol," IETF RFC 5206, 2008.

[16] A. Singh, G. Ormazabal, H. Schulzrinne, Y. Zou, P. Thermos, and S. Addepalli, "Unified heterogeneous networking design," in *IPTComm*, 2013.

[17] A. Yadav and A. Venkataramani, "msocket: System support for mobile, multipath, and middlebox-agnostic applications," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, 2016.

[18] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99, 1999.

[19] H. Balakrishnan and S. Seshan, "Ietf rfc 3124: The congestion manager," 2001.

QAware: A Cross-Layer Approach to MPTCP Scheduling

Tanya Shreedhar*, Nitinder Mohan†, Sanjit K. Kaul*, Jussi Kangasharju†
*Wireless Systems Lab, IIIT-Delhi, India, †University of Helsinki, Finland

***Abstract*—Multipath TCP (MPTCP) allows applications to transparently use all available network interfaces by creating a TCP subflow per interface. One critical component of MPTCP is the scheduler that decides which subflow to use for each packet. Existing schedulers typically use estimates of end-to-end path properties, such as delay and bandwidth, for making the scheduling decisions. In this paper, we show that these scheduling decisions can be significantly improved by including readily available local information from the device driver queues to the decision-making process. We propose QAware, a novel cross-layer approach for MPTCP scheduling. QAware combines end-to-end delay estimates with local queue buffer occupancy information and allows for a better and faster adaptation to the network conditions. This results in more efficient use of the available resources and considerable gains in aggregate throughput. We present the design of QAware and evaluate its performance through simulations, and also through real experiments, comparing it to existing schedulers. Our results show that QAware performs significantly better than other available approaches for all use-cases and applications.**

I. Introduction

Multipath TCP (MPTCP) is a recently-standardized extension to TCP that allows devices with multiple network interfaces, e.g., smartphones with WiFi and LTE, to seamlessly form multiple parallel connections to exploit the full network capacity. MPTCP offers increased robustness and resilience, as well as seamless handovers and it has been proposed to be also used in datacenters [23], opportunistic networks [24], etc. There is both an open source implementation for Linux [21], and companies, such as Apple, have incorporated MPTCP into their products and have made the APIs open to application developers [2].

Figure 1 shows the network stack of MPTCP-compliant machine. Applications utilizing MPTCP can send their data over multiple TCP subflows, where each subflow is associated with a unique network interface. TCP packets scheduled over a subflow wait in the device driver queue of the corresponding network interface before they are transmitted by the network interface card (NIC). The choice of network path for sending application data is made by the MPTCP scheduler block and depends on the scheduling policy.

Scheduling between the multiple connections is an obvious research problem and recently multiple proposals [8], [9], [15], [17] have emerged to improve the default MPTCP scheduler [20]. Typically, these schedulers use a transport layer estimate of the end-to-end bandwidth/delay (for example, the smoothed round-trip time) for each TCP subflow as an input to the scheduling policy that decides on how application data must be assigned to the multiple subflows.

ISBN 978-3-903176-08-9

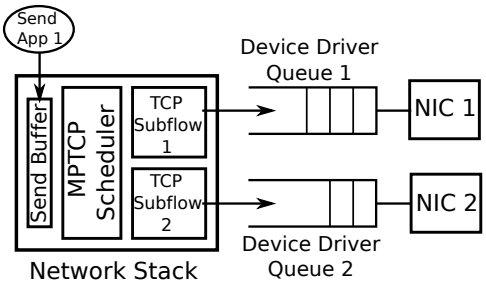


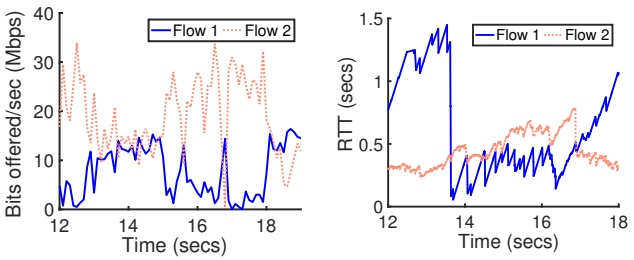
Figure 1: An illustration of MPTCP-compliant machine and how its subflows interact with their corresponding network interface queues.

In this paper, we propose a novel scheduler for MPTCP, QAware, which departs from the previous proposals in a fundamental way. While we also use the end-to-end delay estimates, like current schedulers, QAware additionally considers the number of packets in the device driver queue of the sender. This modification is motivated by our findings, which we discuss further in Section III. The key motivation, as we will demonstrate, is that as a particular flow is used more, its end-to-end delay increases gradually, making it less attractive to use. However, the traditional, purely end-to-end-based estimation, reacts very slowly to these changes.

Additionally, utilizing queue occupancy information allows QAware to use all available subflows optimally, especially when their properties are highly heterogeneous. Existing proposals like [8], [17], [30], treat the flows as separate entities and typically do not fully use all the flows. QAware optimizes transmission over all the flows and gets a significantly higher aggregate throughput, with no loss of performance in any situation.

The contributions of this paper are:

(a) We propose QAware, which is a novel cross-layer approach to scheduling packets across all available MPTCP subflows. The design is motivated by our experimental findings that combining local device driver queue occupancy with the traditional end-to-end delay measurements yield far superior performance.
(b) We model available MPTCP subflows as multiple parallel service facilities that can service data provided by an application. This enables us to leverage queueing theoretical



(a) (b)

Figure 2: (a) Loading (Mbps) at the subflows and (b) their RTT(s). The paths taken by the subflows and the network are shown in Figure 3.

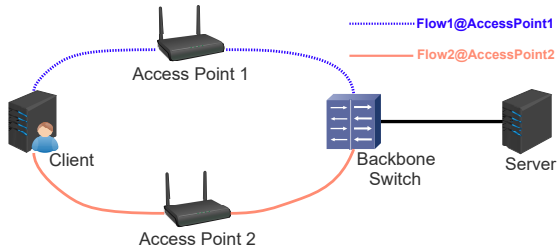


Figure 3: Topology used in experiments and simulations.

insights to create a scheduling policy that combines end-to-end delays and device driver queue occupancy.

(c) Our simulations and real-world experimentation over a wide range of applications compare QAware with the default MPTCP scheduler [20], ECF [17], DAPS [15], and BLEST [8].

Rest of the paper is organized as follows. We discuss the relevant background and related works in Section II. Section III motivates the need for a cross-layer approach to scheduling. In Section IV, we describe the scheduling policy used by QAware. Section V provides implementation details of QAware in latest MPTCP v0.93. Section VI provides an overview of our evaluation methodology. Sections VII and VIII quantify the performance of QAware using extensive simulations and real-world experiments, respectively. We conclude in Section IX.

II. Background and Related Work

The default MPTCP scheduler (minSRTT) allocates traffic on the fastest subflow (one that has the smallest smoothed RTT) with available congestion window at each packet arrival. Several researchers have proposed improvements to the default minSRTT scheduler. Most approaches leverage the difference in RTT of the subflows [3], [11]. Others have also considered additional TCP-layer parameters such as SSThresh, congestion window, selective ACK and receiver buffer size along with RTT [6], [18], [19].

In [30], the authors introduce an additional sender queue to schedule packets on a subflow even when it is unavailable. Delay Aware Packet Scheduler (DAPS) [15] generates a schedule for sending future segments over subflows based on their RTT ratios. However, this makes DAPS unable to react promptly to network changes due to pre-computed long schedules. Blocking-Estimation-based MPTCP Scheduler (BLEST) [8] aims to reduce head-of-line blocking by waiting for the faster subflow despite the availability of space in congestion window of the slower subflow. ECF [17] follows a similar principle as that of BLEST, but while BLEST aims to reduce out-of-order delivery assuming that the send buffer is a bottleneck, ECF aims to minimize completion time.

Researchers have also proposed schedulers that improve MPTCP performance for specific application use-cases. Decoupled Multipath Scheduler (DEMS) [9] aims to reduce fixed-size file's delivery time over MPTCP by estimating available bandwidth on subflows. However, the authors rely on exact knowledge of data chunk boundary for efficient scheduling. In [7] authors leverage application layer information for flow scheduling decisions to provide delay-resilient video streaming in MPTCP. MP-DASH [10] exploits path information from streaming client to improve DASH video delivery. [26] labels WiFi subflow as active/inactive for data transmission based on a minimum desired signal strength. However, unlike other cross-layer approaches which optimize specific application performance over MPTCP, QAware taps into lower layer information to improve performance for *all* MPTCP traffic. Furthermore, as shown later in the paper, QAware's unique design of leveraging hardware queue occupancy enables it to swiftly adapt to varying network conditions and co-existent network applications sharing bottleneck paths.

III. Motivating Use of Cross-Layer Information

Figures 2(a) and 2(b) respectively show loading (bits offered per second) and the corresponding estimates of round-trip times (RTT) of two available subflows by the default MPTCP scheduler, minSRTT. They were obtained from controlled testbed experiments and show how the scheduler optimizes over two available TCP subflows that use non-interfering end-to-end paths. The network topology used in the experiment is shown in Figure 3. The last-mile links were WiFi using 802.11g and the rest were 1 Gbps Ethernet. Neither flow dropped any packets during the length of the experiment.

In the experiment, the default scheduler only utilizes $\approx 60\%$ of available aggregated bandwidth. Observe (Figure 2(a)) that the default scheduler, more often than not, prefers to send packets on one flow over the other. However, this by itself is not responsible for the low utilization of the available bandwidth. The reason, we argue, is that the default scheduler loads a flow deemed to be the best amongst available flows for *undesirably long* intervals. This is because the scheduler uses only the SRTT of the flows, which is a delayed end-to-end transport layer measurement, for its scheduling decisions.

Consider the RTT of flow 1 in Figure 2(b). The RTT captures in a lagged manner the impact of scheduling decision on the subflow. The consistently high values (see interval 12s to 14s in the figure) correspond to an earlier interval of

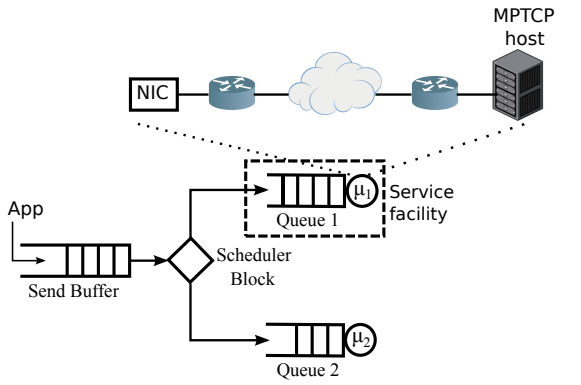


Figure 4: Queueing abstraction of an *end-to-end* MPTCP connection with two subflows.

time when the subflow was being assigned packets by the scheduler while it was heavily loaded. That is, the device queue corresponding to the subflow had previously many packets queued at the NIC.

The sharp dip in values (around time 14s in the figure) captures the transition from when the flow stopped being assigned packets due to high RTT to when it was again assigned packets. These assigned packets arrive at a rather lightly loaded flow and see much smaller RTT, which causes the dip. The small RTT that follows the dip corresponds to packets being assigned to the flow while it was still lightly loaded. As the subflow continues to be assigned packets, the same is reflected, albeit in a delayed manner, in increasing RTT (seconds 16 to 18 in Figure 2(b)) that eventually peaks as it did during $12 - 14$ seconds. By the time the resulting large RTT makes the scheduler switch to the other flow, the scheduler has already spent an undesirably long time injecting packets to a loaded subflow.

In summary, the scheduling decisions that lead to high device queue occupancy and increase in RTT were made using values of RTT that corresponded to an earlier interval when the flow was less loaded. So while a device queue (local to the MPTCP sender and used by the MPTCP flow) is loaded with packets, MPTCP scheduler remains oblivious to the same. Instead, it waits to be informed via a delayed end-to-end RTT based feedback mechanism. In the process, it loses out on many opportunities of scheduling packets to the other better flow; one that is lightly loaded.

The above observations motivate QAware. It uses the occupancy of the device queues together with RTT estimates to use all available flows more efficiently.

IV. QAware Scheduler

We consider a simplified queue-theoretic abstraction to capture the essentials of the scheduling problem, with the goal of maximizing *end-to-end* throughput. Specifically, we model each subflow by a service facility. Figure 4 illustrates the abstraction for an MPTCP *end-to-end* connection that uses two TCP flows. The abstraction allows us to apply results from analysis of multi-queue systems [25].

In our queueing abstraction, packets generated by an application arrive into a queue that models the TCP send buffer (Figure 1). Packets in this queue are assigned to one of the available service facilities in a first-come-first-serve (FCFS) manner. Each facility consists of a finite queue and a server. Packets inside a facility are serviced in an FCFS manner.

The queue in a service facility is the device driver queue (Figure 1) that is used by the TCP subflow corresponding to the facility. The server includes the source host NIC, access network used by the subflow, intermittent nodes in the core and the destination host (all layers of the TCP/IP stack).

When a packet is assigned to a service facility, it may find other packets waiting for service in the facility's queue. This packet must wait for all the other waiting packets to finish service before it enters the server of the facility. The total time a packet spends in a facility, often referred to as its *system time*, includes the time it waits in the facility's queue and the time it spends getting service.

Origins of the QAware scheduler: Many analytical works on queueing systems have looked at scheduling customer/packet arrivals to parallel service facilities [25], [27]–[29]. For many general arrival processes and service time distributions, when all servers are stochastically identical, the optimal policy is to choose a service facility with a minimum number of packets in its queue [25], [27], [29], that is it minimizes the average packet system time. For the case of non-identical servers, a scheduling policy that assigns a packet to a service facility that minimizes the conditional expected system time of the packet, conditioned on the knowledge of the number of packets waiting for service in the facility, shows good performance [25]. Our QAware scheduler uses the policy in an MPTCP setting.

Consider K service facilities indexed $1, \ldots, K$. Let facility k have a service rate of μ_k. The two facilities in Figure 4 have service rates of μ_1 and μ_2. Let $n_k(t)$ be the number of packets waiting for service in facility k at time t. The policy assigns a packet to a service facility k^* given by

$$k^* = \arg\min_k \frac{n_k(t) + 1}{\mu_k}. \tag{1}$$

Note that $1/\mu_k$ is the expected service time of a packet in facility k. Thus, the conditional waiting time of a packet that enters such a facility is $n_k(t)/\mu_k$, which is the sum of the expected service times of the $n_k(t)$ packets currently waiting for service in the facility. In addition, we add the term $1/\mu_k$ to $n_k(t)/\mu_k$, to include the expected service time of the packet to be scheduled. Thus, the expression being minimized in (1) is the conditional expected *system time* of a packet if it were to be assigned to facility k.

Adapting scheduling policy (1) to multiple end-to-end TCP subflows: The number $n_k(t)$ of packets in the queue of service facility k is the number of packets waiting in the device driver queue of the corresponding subflow k and can be obtained.

However, we must estimate the average service time $1/\mu_k$ of subflow k.

Consider the i^{th} packet arrival. Let t_i^s be the time the packet is assigned to a subflow. Let t_i^a be the time that a TCP ACK acknowledges receipt of the packet. The round-trip time of the packet is $\text{RTT}_i = t_i^a - t_i^s$. Note that this includes the time packet waits in the device driver queue of its assigned subflow before it starts service and the time it spends in service. This is the *system time* of the packet. Let W_i[1] be the time the packet i waits in the queue. This time can be calculated locally at the MPTCP sender. The time X_i that the packet spends in service begins when the packet enters the NIC for transmission and ends when a TCP ACK for the packet is received. Given W_i and RTT_i, we have $X_i = \text{RTT}_i - W_i$. The estimate of the service time is updated on receipt of a TCP ACK. Let $\hat{S}_k$ be the current estimate of the average service time of facility k. On receipt of a TCP ACK for packet i, we update

$$\hat{S}_k = \alpha \hat{S}_k + (1-\alpha) X_i, \tag{2}$$

where $0 < \alpha < 1$ applies appropriate weights to the last estimate of the average and the current service time. We use $\alpha = 0.8$ in this work which is also the smoothing factor for TCP congestion control [2] . The corresponding estimate of the service rate is $1/\hat{S}_k$. At time t, QAware schedules to the TCP subflow k^* that satisfies

$$k^* = \arg\min_k (n_k(t) + 1)\hat{S}_k. \tag{3}$$

Finally, note that since $X_i = \text{RTT}_i - W_i$, we have $\hat{S}_k = \text{RTT} - \widehat{W}$, where RTT and $\widehat{W}$ are the exponentially weighted moving averages, with coefficient α, of packet round-trip times and device driver queue waiting times, respectively, for the subflow k. In our real implementation, summarized in (Algorithm 1), we use RTT estimates that are readily available for each subflow and we calculate an approximation of $\widehat{W}$ based on information available from device driver queues.

V. Implementation

We implement QAware as a modular scheduler using MPTCP v0.93 based on Linux kernel v4.9.60 [12]. The code is available at [22].

As shown in Section IV, QAware's functioning depends on the current estimate of network interface (NIC) queue occupancy. Conventionally, the NIC queues were either implemented within the hardware itself or as part of the driver; which made NIC queues invisible to the Kernel and its occupancy extremely hard to estimate. However, since Linux Kernel $>$ v3.3.0, several NIC queue management protocols, known as Byte Queue Limits (BQL), have been introduced as part of the Kernel code to resolve starvation and latency at the NIC [14]. The BQL algorithms push queueing abstractions from hardware drivers to specific data structures which can be accessed from within the Kernel [3].

Algorithm 1 QAware Algorithm

```
Inputs:
    Available Subflows SF∈ {1, ..., n}
Initialize at packet arrival P_k:
    minService ← 0xFFFFFFFF
    selectedSubflow ← NONE
//The function below will return best subflow for packet P_k
for each subflow ∈SF do
    n_k ← queueSize(subflow)
    if n_k ≠ 0 then
        Δt ← sampling time
        Δpackets ← packets dequeued in Δt
        W_k ← [1/(Δpackets/Δt)]n_k
    else
        W_k ← 0
    end if
    Ŵ ← αŴ + (1 − α)W_k
    Ŝ_k = [RTT − Ŵ]
    TS_k = (n_k + 1)Ŝ_k
    if TS_k < minService then
        minService ← TS_k
        selectedSubflow ← subflow
    end if
end for
```

Our implementation closely follows the Algorithm 1. We first tap the network device address mapped to MPTCP socket via `struct dst_entry` to access *DQL*[4] as follows:

```
dql = netdev_get_tx_queue(dst->dev)->dql
```

We further utilize *DQL* entry to estimate current NIC (*netdevice*) queue occupancy of each MPTCP subflow.

```
qSize = {dql->num_queued -
         dql->num_completed}
```

Here, `num_queued` and `num_completed` refer to the total number of bytes queued in the network device and number of bytes successfully transmitted by the device respectively.

Apart from NIC queue estimates, we utilize the smoothed mean RTT estimates in microseconds via `srtt_us` accessible through `struct tcp_sock`. We ensure that our implementation is in line with guidelines mentioned in RFC 6182 [13].

VI. Evaluation Methodology

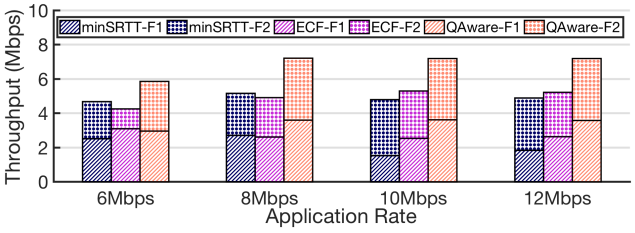
In following sections, we evaluate QAware's performance through an extensive set of simulations and real-world experiments. We model our evaluation methodology to mimic real MPTCP network configurations and application use-cases. In majority of our evaluation, we model a realistic network scenario (as illustrated in Figure 3) wherein a client leverages two distinct network paths to connect to a distant server.

[1]For simplicity of exposition we ignore the time a TCP ACK may have to wait in a queue before being sent to the TCP layer.

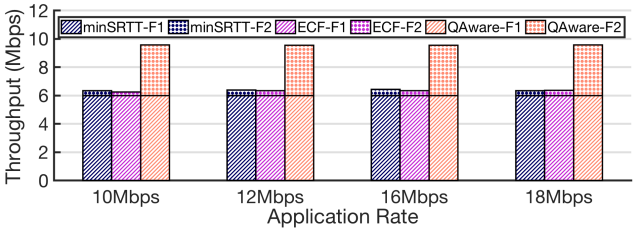
[2]We examined for other values of α which did not impact the overall performance of QAware.

[3]Currently, only PCIe-based ethernet drivers support BQL [5]. However, a significant effort is being made from the Linux developer community to support broader list of NICs, including wireless NIC's [4].

[4]In Linux, BQL is implemented as Dynamic Queue Limit (DQL).

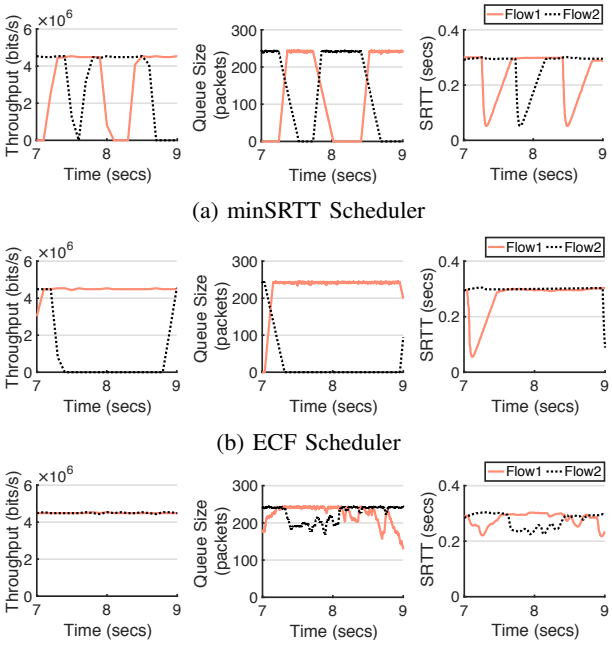


(a) Subflows F1 and F2 use links with PHY rates of 6 Mbps.



(b) Subflow F1 and F2 use links with PHY rate of 12 Mbps and 6 Mbps respectively.

Figure 5: Throughput achieved by minSRTT, ECF and QAware schedulers for different CBR rates.



(a) minSRTT Scheduler

(b) ECF Scheduler

(c) QAware Scheduler

Figure 6: Per-flow throughput, device driver queue occupancy, and SRTT behavior as a function of time. These correspond to the throughputs in Figure 5(a) and a CBR rate of 12 Mbps.

For simulations, we implement QAware on ns-3 network simulator. We compare QAware with default minSRTT and Earliest Completion First (ECF) [17] scheduler for constant bit rate (CBR), file downloads, and web browsing workloads. The simulations help us zoom into the workings of the schedulers and allow us to evaluate QAware over a variety of workloads and network path configurations. Our evaluation setup and results are described in Section VII.

We further examine and validate the performance gains obtained by QAware in simulated environments via real network experiments. We utilize our Kernel implementation summarized in Section V. The experiments were performed in a university data center and consider a variety of workloads such as video streaming, web file downloads, etc. We compare QAware with several state-of-the-art schedulers such as minSRTT, Delay Aware Packet Scheduler (DAPS) [15], Blocking Estimation based scheduler (BLEST) [8], and ECF [17]. The details of our experiments and consequent results are discussed in Section VIII. All our results throughout evaluation are averaged over multiple runs.

VII. Simulation Setup and Results

We simulated network topologies of the kind shown in Figure 3. For all simulations, the links between the access points and the backbone switch and between the backbone switch and the server were modeled as wired links with rate 30 Mbps and 50 Mbps respectively. The client is connected to the two access points over wireless links with physical layer (PHY) rates in the range $6-12$ Mbps. These two wireless links provided the two network paths over which application data was sent. Both subflows use independent congestion control.

We simulated the following *applications*: i) constant bit rate (CBR) data from low to high rates, ii) file transfer for sizes of $10-30$ MB, iii) web browsing of top 10 out of the US Alexa-100 websites, and iv) CBR with one of the paths being shared by UDP traffic. For the applications, we simulated the following *network configurations*: i) both wireless links have the same rate, ii) one link is much faster than the other, and iii) one link drops TCP packets. Comparisons of QAware with ECF and minSRTT[5] demonstrate the benefits that are accrued by QAware because it optimally utilizes both network paths.

A. Constant Bit Rate Traffic

Access paths with no packet errors: Figure 5(a) shows the TCP throughputs obtained by the schedulers for increasing CBR rates. Each wireless link was configured with a PHY rate of 6 Mbps. This results in homogeneous network paths. On average, QAware achieves percentage throughput gains of about 40% over the rest. Further, note that all schedulers use both subflows. However, unlike the others, QAware utilizes both the subflows almost equally for the entire simulation time for all the CBR loads. To better understand their behaviors, consider Figure 6, which shows for each scheduler and subflow, the variation of throughput, device driver queue occupancy, and smoothed RTT, as a function of time, for a

[5]In simulation, the scheduler assigns packets over independent TCP streams. We do not incorporate other MPTCP functionality such as retransmission handler and path manager.

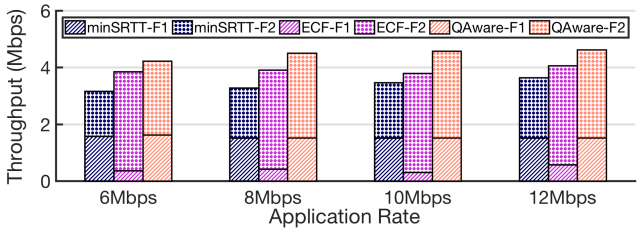


Figure 7: Per-flow throughput comparison for different CBR rates where subflow F1 experiences a packet drop rate of 10^{-2}.

2 second interval. The CBR rate was set at 12 Mbps. From the subflow throughputs and queue occupancy, it is clear that QAware uses both subflows almost simultaneously. ECF uses just one subflow for most of the interval, and while minSRTT uses both flows during the interval, it switches between them very infrequently. Both minSRTT and ECF rely on the delayed feedback provided by SRTT and so end up scheduling packets to one subflow for longer intervals than QAware. Essentially, they switch flows when SRTT of the subflow in use exceeds that of the other subflow. In addition, ECF, by design, declines scheduling opportunities to a subflow with a larger RTT and prefers to wait for faster subflows. This explains the reason for using one flow for a longer duration than minSRTT scheduler. In minSRTT and ECF, subflows experience swings in SRTT. The SRTT increases linearly while it is the subflow of choice. This increase eventually makes the subflow less desirable than the other and the scheduler switches to the other flow, which, due to the current low occupancy in the corresponding device queue, experiences low SRTT.[6]

Figure 5(b) shows throughputs obtained by the CBR application when the PHY rate of one of the wireless links is 6 Mbps and the other is 12 Mbps. While all schedulers utilize the subflow using the 12 Mbps link equally, QAware also utilizes the subflow mapped on the 6 Mbps link. On average, QAware achieves throughput gains of about 50% over the rest.

Access paths with packet errors: Figure 7 shows the throughput obtained when one subflow suffers a packet loss rate of about 10^{-2}. Both wireless links have PHY rates of 6 Mbps. Upon detecting packet loss, the congestion window of the subflow decreases based on *TCP congestion avoidance algorithm*, which limits the number of packets that can be sent on that subflow. Even in this situation, QAware is able to exploit both subflows better and achieves about 32% and 15% improvement over minSRTT and ECF respectively. For the case when the wireless links are 12 Mbps and 6 Mbps with an error on the slower link, the corresponding gains are 53% and 6% (figure not shown due to space limitations). Note that since ECF is biased toward using the faster path, it performs almost as well as QAware when the error-free path has a faster wireless link. On the other hand, while minSRTT uses the error-prone path better than ECF, it is unable to make good use of the error-free path as the other two schedulers.

[6]Our observations with respect to QAware and minSRTT for three homogeneous paths are similar. We skip them due to lack of space.

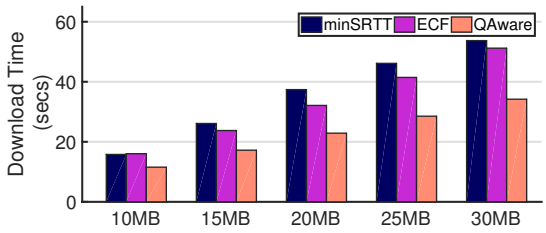


Figure 8: File download completion times when both subflows use wireless link with PHY rate of 6 Mbps.

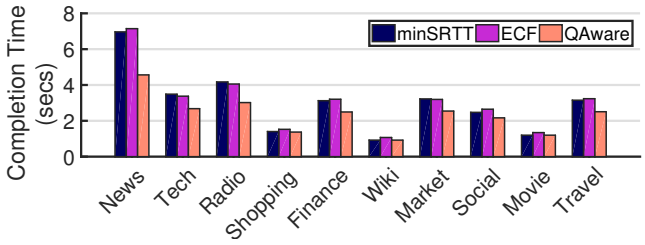


Figure 9: Download completion time for 10 websites from top U.S. Alexa-100 websites.

B. Fixed Size File Transfer

Figure 8 shows the download completion time achieved by the three schedulers for five different file sizes ranging from 10MB to 30MB. Both wireless links were set to a PHY rate of 6 Mbps. Observe that QAware obtains the least download time for all the file sizes. This is explained by its ability to effectively utilize both the subflows for data transfer. The performance gap increases proportionally with file size. Overall, QAware achieves 35% and 30% reduction in average download time over minSRTT and ECF respectively.

C. Web-browsing

To simulate web browsing, we deployed objects of 10 out of top U.S. Alexa-100 websites, which are summarized in Table I, in our simulated server. The client consecutively downloaded relevant objects of each website from the server at a variable rate between 10Mbps to 30Mbps chosen in a probabilistic manner. We compared scheduler performance for when both wireless links are 6 Mbps and when one of the links is 12

Website	News	Tech	Radio	Shopping	Finance
#Objects	202	67	66.2	52.2	39.7
Size (KB)	3821.2	2152.2	2453	1000.7	1988.1
Website	Wiki	Market	Social	Movie	Travel
#Object	28	49	69	39	21
Size (KB)	601.2	2032.8	1700.2	845.7	2000.4

Table I: Web objects for traffic generation

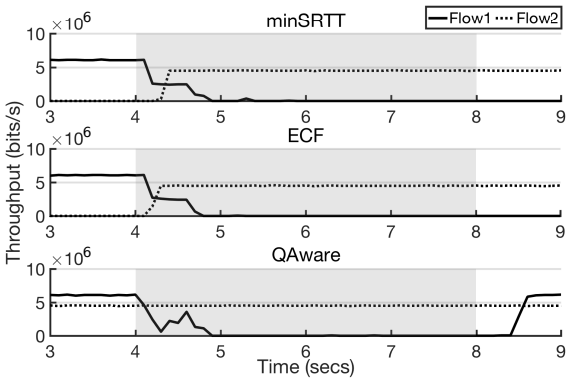


Figure 10: Per-flow throughputs when the interface used by subflow F1 sees UDP traffic for 4 seconds (greyed).

Mbps. QAware achieves a significant reduction in download completion time for both configurations, specifically up to 35% for the former (see Figure 9) and up to 28% for the latter (figure not shown due to space limitations). On the other hand, ECF and minSRTT perform similarly.

D. Multiple Applications

In current computing environments, end hosts typically run multiple applications which must share the interfaces available at the host for network transfers. An ideal MPTCP scheduler must be able to efficiently adapt to bandwidth competition on bottleneck links in such coexisting environment. To evaluate the impact of such sharing on the schedulers, we used the following setup. The PHY rates of the wireless links were set to 9 and 6 Mbps. A CBR application generated data for a 10 second interval and used both the MPTCP subflows. The results are shown in Figure 10.

Starting at 4 seconds, we introduced traffic from a UDP application that used the network path with the 9 Mbps wireless link. The greyed area in the figure denotes the time duration when both MPTCP and UDP applications were active at the client. The UDP traffic lasted for 4 seconds. Before the start of the UDP traffic, only QAware scheduler was utilizing both available subflows. Once the UDP application starts, the device queue of the 9 Mbps wireless link saturates. QAware, however, quickly adapts to it and reduces the traffic being sent on the corresponding subflow. All the while, it keeps utilizing the subflow over the slower wireless link. On the other hand, both minSRTT and ECF need to wait for several RTT updates for the impact of UDP traffic on queue wait times to get reflected in the SRTT of the subflow. Lastly, unlike the other schedulers, QAware is also quick to detect the availability of the subflow after the 8 second mark, which is when the UDP application stops its transfer. Overall, QAware leads to gains of about 40% over minSRTT and about 50% over ECF.

VIII. Real-World Setup And Experiments

We next examine QAware's performance in real network environments. Figure 11 shows our test network topology

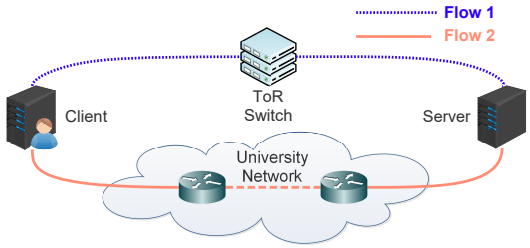


Figure 11: Real network testbed in university datacenter.

in University of Helsinki data center. We assign two similar machines with 16 core AMD Opteron processor, 8 GB DDR2 RAM running Ubuntu 16.04 LTS with latest stable MPTCP implementation (version 0.93, based on Linux kernel v4.9.60 [12]) as client and server. The implementation uses default congestion control algorithm (coupled OLIA). Both machines are interconnected via two separate Gigabit Ethernet interfaces. One Ethernet connection is routed through internal University of Helsinki network and therefore encounters background traffic from University staff. It has an end-to-end RTT of >1ms. The other connection is over Top-of-Rack (ToR) switch with RTT <1ms.

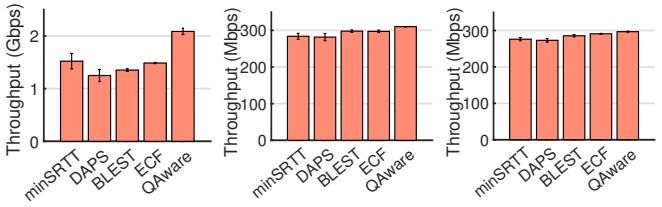
We compare QAware with the following schedulers: i) minSRTT, ii) Delay Aware Packet Scheduler (DAPS) [15] iii) Blocking Estimation based Scheduler (BLEST) [8], and iv) Earliest Completion First (ECF) [17][7] [8]. We first compare scheduler performance for application generating bulk traffic. This workload provides a qualitative validation of the results we obtained in Section VII. We further present scheduler performance for DASH video streaming and web file downloads. We used the Linux Traffic Control system (*tc*) in combination with a Hierarchical Token Bucket (HTB) packet scheduler using Statistical Fair Queuing (SFQ) for network shaping. In between runs, we flushed out the TCP cache to ensure that each run is independent of the next. All our results are averaged over ten runs.

A. Bulk Traffic

In this section, we compare QAware's performance with other schedulers for high application transfer rate over both subflows. We performed experiments with different settings of delays along the two paths. The setting includes i) default path delays (< 1ms and > 1ms), ii) delay shaping to introduce 40ms of delay along one path and 80ms along the other, and iii) 40ms along one path and 160ms along the other. Path bandwidths corresponding to the different delays are stated in Table II(a).

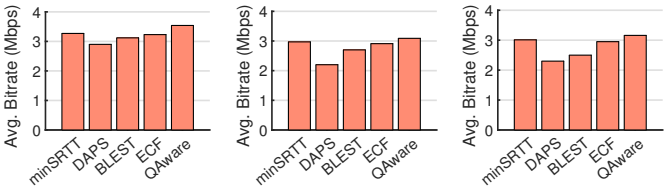
[7]For DAPS and BLEST, we use the implementation at https://bitbucket.org/blest_mptcp/nicta_mptcp. For ECF, we use the implementation at http://cs.umass.edu/~ylim/mptcp_ecf

[8]DAPS, BLEST, and ECF are implemented on MPTCP v0.89 whereas the default minSRTT and QAware are based on MPTCP v0.93. We could not implement QAware on MPTCP v0.89 as it is based on Linux v3.18 which does not support BQL. Please see [12] for exact changes between the two versions.



(a) Default (b) 40+80ms (c) 40+160ms

Figure 12: Bulk Traffic throughputs for different access path delays.



(a) 2+2 Mbps (b) 2+1.6 Mbps (c) 2.4 +1.6 Mbps

Figure 13: Average bitrate in video streaming for different path bandwidths.

Figure 12(a) compares average throughput obtained by different schedulers for default path delays. QAware achieves more than 45% increase in throughput compared to DAPS, BLEST and ECF. QAware also provides an improvement of 37% over the default minSRTT scheduler. Interestingly, the minSRTT scheduler outperforms DAPS, BLEST, and ECF in the experiment. We attribute minSRTT's efficiency to two reasons. Firstly, DAPS, BLEST and ECF schedulers have been designed to improve MPTCP performance for heterogeneous delays along available network paths. In fact, BLEST and ECF even go as far as not sending an available packet on a slower subflow and wait for the faster subflow to become available. When subflows witness similar delays (as in the current case), the default scheduler places more packets on each path as opposed to DAPS, BLEST, and ECF. Secondly, based on latest MPTCP kernel, minSRTT enjoys several code improvements and optimizations.

For when the path delays are 40 and 80ms, QAware yields an average throughput of 310 Mbps which is an improvement of about 10% over the default scheduler and DAPS and 5% over ECF and BLEST (shown in Figure 12(b)). As presented in Figure 12(c), all schedulers perform quite similar to each other as all try to fully utilize the lower delay subflow when path delays are 40 and 160ms. In this case, QAware still manages to achieve an improvement of about 7% over the default scheduler and DAPS, and about 4% over BLEST and ECF.

B. Video Streaming

Streaming is a dominant Internet use case and is widely adopted by content providers such as Netflix and YouTube [1]. We set up a DASH server and host *Big Buck Bunny*, available from a public dataset, on it [16]. We configured the streaming

Delay (ms)	1	40	80	160
Bandwidth (Mbps)	950	600	300	200

(a) Configurations for Bulk Traffic Experiments

Bandwidth (Mbps)	2.4	2	1.6
Delay (ms)	10	20	30

(b) Configurations for Video Streaming Experiments

Table II: II(a) shows bandwidth achieved by delay throttling on a 1Gbps Ethernet interface whereas II(b) presents values after both bandwidth and delay shaping

server to provide five representations of the video from 240p to 1080p (same as most content providers). We re-encoded each representation in at least three different bitrates with overall available bit rates from 128Kbps to 3.8Mbps using H.264/MPEG-4 AVC codec. The streaming client employs an Adaptive Bit Rate (ABR) algorithm to download video segments according to the available network bandwidth. We throttled our testbed bandwidth to match the bitrates of DASH encodings. Table II(b) shows the average delay measured at client-side for each bandwidth configuration. We evaluate and compare QAware's performance with other schedulers for when the two subflows i) have bandwidths of 2 Mbps, ii) have bandwidths of 2 Mbps and 1.6 Mbps, and iii) have bandwidths of 2.4 Mbps and 1.6 Mbps.

From Figure 13, we observe that QAware improves the performance of streaming applications in all network conditions. The performance improvement is more significant in scenarios where the path bandwidths are similar (8% and 5% with respect to default and 10% and 6% with respect to ECF, in Figures 13(a) and 13(b) respectively) as QAware utilizes available paths more efficiently than other schedulers. DAPS consistently gives the worst performance out of all schedulers due to its strong dependence on RTT ratio of two subflows.

C. Web File Download

We now evaluate QAware's performance for simple web downloads using *curl*. We set up an HTTP server using Apache 2.2.22 and hosted varying file sizes of range 128KB to 500MB. We eliminate application connection time by only considering the transport-level time in overall download completion time observed at the client. Figure 14 presents the average com-

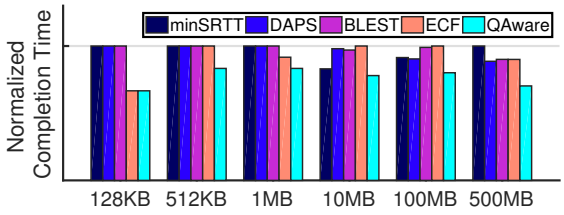


Figure 14: Normalized download completion time for different file sizes (smaller is better).

pletion time normalized to the maximum achieved value by scheduler for a given file size.

For small web transfers (<1MB) all schedulers perform quite similar to each other (it took 0.002s to download a 128 KB file by QAware vs. 0.003s by minSRTT). This is because for small data transfers, the bandwidth of the primary subflow is more than capable of single shot transmission and thus MPTCP rarely switches to the secondary subflow. Therefore, until the performance of primary subflow degrades during transfer, the choice of the scheduler does not affect the performance for small files. The default and DAPS scheduler achieve lower completion time for medium file sizes (≈10/100 MB) in comparison to BLEST and ECF. This is likely because BLEST and ECF add additional delays by waiting for the faster subflow to become available. For large files (500 MB), BLEST and ECF utilize faster subflow more efficiently than default and DAPS, thus achieving a lower completion time. QAware always outperforms other schedulers and realizes up to 20% decrease in completion time for medium file sizes (0.709s by QAware vs. 0.895s by ECF for 100 MB file) and 30% for large file downloads (3.46s by QAware vs. 4.93s by minSRTT for 500 MB).

IX. Conclusion

We proposed, QAware, a novel cross-layer MPTCP scheduler that combines hardware device queue occupancy and TCP RTT for efficient scheduling decisions. We detailed its design and implementation. We evaluated QAware using an extensive set of simulations and real network experiments for various network configurations and applications such as bulk data transfers, web browsing, web file downloads, and video streaming. Comparisons with various state-of-the-art schedulers such as DAPS, BLEST, and ECF were used to demonstrate the efficacy of QAware. It outperformed other schedulers in all network configurations and workloads we tested. Further, we show that QAware quickly adapts to co-existing applications and sudden variations in network conditions. We have open-sourced QAware's implementation as a modular scheduler for latest stable MPTCP Linux release.

Acknowledgment

This research was funded by TCS Research Scholarship Program, EU FP7 Marie Curie Actions Cleansky Project (Contract No. 607584) and Young Faculty Research Fellowship (Visvesvaraya Ph.D. scheme) from MeitY, Govt. of India.

References

[1] Global Internet phenomenon. www.sandvine.com/.../global-internet-phenomena-report-latin-america-and-north-america.

[2] Apple Inc. Use Multipath TCP to create backup connections for iOS. https://support.apple.com/en-us/HT201373, 2017.

[3] S. H. Baidya and R. Prakash. Improving the performance of multipath tcp over heterogeneous paths using slow path adaptation. In *2014 IEEE International Conference on Communications (ICC)*, 2014.

[4] Bufferbloat community. Make WiFi fast project. https://www.bufferbloat.net/projects/make-wifi-fast/wiki/, 2014.

[5] Bufferbloat community. The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm. https://tools.ietf.org/id/draft-ietf-aqm-fq-codel-06.html, 2014.

[6] Y. Cao, Q. Liu, G. Luo, and M. Huang. Receiver-driven multipath data scheduling strategy for in-order arriving in sctp-based heterogeneous wireless networks. In *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015.

[7] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. Cross-layer scheduler for video streaming over mptcp. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16.

[8] S. Ferlin, . Alay, O. Mehani, and R. Boreli. Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, 2016.

[9] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen. Accelerating multipath transport through balanced subflow completion. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, 2017.

[10] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. Mp-dash: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on CoNEXT*, 2016.

[11] J. Hwang and J. Yoo. Packet scheduling for multipath tcp. In *Seventh International Conference on Ubiquitous and Future Networks*, 2015.

[12] M. T. IETF. MPTCP Linux implmentation v0.93. http://multipath-tcp.org/pmwiki.php?n=Main.Release93, 2017.

[13] Internet Engineering Task Force (IETF). Architectural Guidelines for Multipath TCP Development. https://tools.ietf.org/html/rfc6182, 2011.

[14] Internet Engineering Task Force (IETF). BQL enabled drivers. https://www.bufferbloat.net/projects/bloat/wiki/BQL_enabled_drivers/, 2014.

[15] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. Daps: Intelligent delay-aware packet scheduling for multipath transport. In *IEEE International Conference on Communications (ICC)*, 2014.

[16] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over http dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, 2012.

[17] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *Proceedings of the 13th International Conference of CoNEXT*, 2017.

[18] D. Ni, K. Xue, P. Hong, and S. Shen. Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath tcp in lossy networks. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014.

[19] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu. Ocps: Offset compensation based packet scheduling mechanism for multipath tcp. In *2015 IEEE International Conference on Communications (ICC)*, 2015.

[20] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, 2014.

[21] C. Paasch and B. Sebastian. Multipath TCP in the Linux Kernel. http://www.multipath-tcp.org, 2017.

[22] QAware. QAware scheduler for MPTCPv0.93. https://github.com/nitinder-mohan/mptcp-QueueAware.git, 2018.

[23] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 Conference*.

[24] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley. Opportunistic mobility with multipath tcp. In *Proceedings of the Sixth International Workshop on MobiArch*, MobiArch '11, 2011.

[25] Z. Rosberg and P. Kermani. Customer routing to different servers with complete information. *Advances in Applied Probability*, 21, 1989.

[26] Y. s. Lim, Y. C. Chen, E. M. Nahum, D. Towsley, and K. W. Lee. Cross-layer path management in multi-path transport protocol for mobile devices. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014.

[27] R. R. Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 1978.

[28] W. Whitt. Deciding which queue to join: Some counterexamples. *Oper. Res.*, 1986.

[29] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 1977.

[30] F. Yang, Q. Wang, and P. D. Amer. Out-of-order transmission for in-order arrival scheduling for multipath tcp. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014.

The effect of network topology on the control traffic in distributed SDN

Muhammad Zeshan Naseer and Viktoria Fodor
Department of Network and System Engineering, KTH - Royal Institute of Technology, Stockholm
{mznaseer, vjfodor}@kth.se

***Abstract*—Software Defined Networking (SDN) has the promise of flexible routing, traffic management and service provisioning in communication networks. To allow SDN based networks scale in size, the control architecture needs to be distributed, which in turn requires the introduction of controller to controller communication. This is needed to ensure that the distributed controllers have the same understanding about the underlaying network and can make consistent local decisions. In this paper we evaluate the volume of the emerging control traffic, considering a distributed controller architecture based on ONOS and OpenFlow. We show that the control traffic increases drastically with the number of controllers, as well as with the size of the underlaying network. We evaluate topologies forming regular and random graphs, and conclude that the type of the topology influences the traffic volume significantly, while the network density has less significant effect. We show that the control traffic is significant even if the number of controllers is selected such that the control traffic is minimized, and we argue that further optimization of ONOS is needed to trade off control traffic load and consistency in the network views.**

***Keywords*—SDN, ONOS, control, scalability**

I. Introduction

Software Defined Networking (SDN) is becoming a generally accepted solution to provide the increased flexibility needed for service differentiation and resource efficiency in wired as well as in wireless networks [1] [2]. In an SDN, a centralized controller takes over the control from the individual switching nodes. The network operating system collects the information about the network, and helps the controller to make an abstract model of the network topology. This complete knowledge of the network helps the controller to dynamically provision the network resources, to apply the concepts of fairness and traffic shaping, as well as to provide complex routing policies for enhancing security, achieving quality of service differentiation or to support network function visualization.

The use of a single controller however raises reliability and scalability issues. First, a single controller SDN becomes a single point of failure, which is critical for network performance and reliability. Second, a single controller cannot support a large network due to limited memory and processing capabilities. Finally, the position of a single controller is also critical in terms of switch to controller network delays. As a result, a network based on a single controller would have scalability constraints based on memory, processing capability and reaction time.

ISBN 978-3-903176-08-9 2018 IFIP

For improved scalability of the network, a logically centralized controller can be implemented through a cluster of controllers, leading to a distributed SDN architecture. This architecture permits to balance the switch to controller traffic among different controllers, improves reliability and can achieve scalability by limiting the load of a single controller and the switch to controller network delays. However, this architecture also implies that the controllers must coordinate to obtain a consistent view of the network state [3]. As the set of controllers need to share information about the switches they own and about the network topology, controller to controller traffic is introduced which may be non-negligible as the number of controllers or the network itself grows [4].

The objective of this paper is to evaluate the effect of the network topology on the emerging control traffic, and to discuss how these traffic can be minimized. We estimate the volume of the emerging control traffic in large distributed SDNs, that applies ONOS for controller to controller and OpenFlow for switch to controller communication. We utilize the measurement results of [4], and build a model for the controller to controller traffic for general network topologies. Then we consider the specific cases of regular and random network topologies, and evaluate the effect of the network size and of the number of controllers. We show that for the same network size and density, the control traffic is higher in regular networks, but the scalability properties in the two topologies are similar. The optimal number of controllers depends significantly on the network size and on the intensity of the switch to controller queries, while the density of the network has only marginal effect.

II. Related work

The placement of controllers is one of the main issues in the design of distributed SDNs and the literature addresses controller placement for various objectives. The seminal paper [5] proposes controller placement heuristics that trade off latency, failure tolerance and load balancing. In [6], [7] a given set of controllers is placed, such that the computational capacity of the controllers is obeyed and the maximum switch to controller delay is minimized, while [8] minimizes the cost of controller deployment and maintenance, keeping the constraints on the controller and network capacities as well as

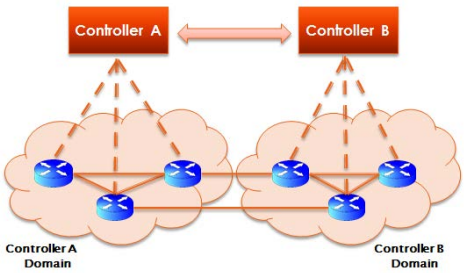


Fig. 1. Distributed SDN architecture. Controllers A and B exchange information to form a virtual centralized controller.

on the switch to controller delays. Reliable distributed SDN is considered in [9], [10], where delay and capacity constraints have to be fulfilled even when backup controllers are used.

There are only few works addressing the controller placement problem from the point of view of the control traffic, since network operating systems for distributed architectures are in their infancy. The emerging ONOS is evaluated in [4], by emulating small networks with the objective of finding mathematical models of the controller to controller traffic load. These findings are then used in [11] to evaluate the effect of controller placement on the control traffic in given, large topologies. In our work we build as well on the measurement results of [4] to systematically evaluate how the control traffic is affected by the network size and network topology. To estimate the rate of switch to controller traffic for flow setup, we use the measurement results of [12]. More exact models considering network topology as well as traffic matrix are also available in the literature [13], these could be considered for more detailed studies.

III. Distributed SDN

In this paper we consider a distributed software defined network with the general architecture shown on Figure 1. According to the SDN principles, the control functions are decoupled from the forwarding functions at the network switches, and are implemented by a set of controllers, each responsible for a cluster of switches. The controllers exchange information to have a common network view and to form a virtually centralized controller.

In this architecture two control planes can be identified: the switch to controller (s2c) plane , which is present already in a centralized SDN and supports the set up of the forwarding tables based on the transmission paths determined at the controller, and the controller to controller (c2c) plane, which ensures correct controller functionalities despite the distributed implementation. The availability of the shared data structures affects the process of providing information for the switches querying their controller. In this paper we consider the multiple data-ownership model, where each controller has a local copy of all data required for routing decisions, therefore switch queries never need to be forwarded in the c2c plane.

The s2c plane has rich literature with mature architecture designs and protocol implementations like OpenFlow [14]. The development of the c2c plane is less mature, where ONOS [15] and OpenDaylight [16] are two prominent open implementations. In this work we build our analysis on the characteristics of ONOS, but our results could be generalized for other solutions.

The design of the c2c plane needs to address two main issues: first, the consistency of the shared view of the network graph, which is required for the correct control of the forwarding plane; and second, the availability of the shared data structures, required for fast decision making. Consistency in general can be strong or eventual. Strong consistency means that all available information is identical at all the controllers, while eventual consistency means that the controllers may have temporally different views, but these will converge with time. In systems with communication delays strong consistency compromises all time data availability [17]. Therefore, ONOS combines strong and eventual consistency models, to trade-off between delay and consistency:

1) The controllers need to have an agreement all the time on the assignment of the switches. Therefore, cluster membership is shared under the strong consistency model, using the Raft protocol [18]. Raft messages are exchanged when the membership of a cluster has changed – that is, a new switch got connected or a switch has left.
2) Other information, including the network topology, is shared under the eventual consistency model, using the so called anti-entropy algorithm [19], implementing simple gossiping among all the controllers. In the anti-entropy algorithm, controllers send out update messages periodically to randomly selected controllers and in this way the information on the changes eventually propagates in the network of controllers. The consensus time for the fully connected network of C controllers is $O(\ln C)$ [20].

IV. Control traffic model

A. Methodology

Our objective is to build up an SDN control traffic model, including both controller to controller (c2c) and switch to controller (s2c) traffic. Since control messages are transmitted through the network links over multihop paths, the model need to reflect the length of the transmission paths across the network that the control messages travel. Therefore, we derive the control traffic load in two steps: first we express the bandwidth (or rate) of the generated c2c and s2c control traffic, B_{c2c} and B_{s2c} and then weight them by the length of the transmission paths the control messages travel, to get the total control traffic $T_T = T_{c2c} + T_{s2c}$.

Our objective is to evaluate the effect of the topology on the control traffic load. Therefore, we consider two complementary topologies, regular grids and Erdős-Rényi random graphs. These graph structures represent two extremes in the

world of network models: regular grids have large average path length, but also high clustering coefficient. On the other end, ER graphs have logarithmic increase in path length, but low clustering coefficient. These graph metrics are important in the case of distributed SDNs, for efficient controller to controller and switch to controller communication respectively.

For simplicity, throughout the paper we assume that fraction and square root operations give integer values. If it does not hold, the results are approximate.

B. Control traffic bandwidth

We consider a network of S switches and C controllers, forming a clustered distributed SDN architecture, where each controller is responsible for a subset of the switches.

The ONOS control traffic bandwidth have been measured systematically for networks with two and three controllers in [4]. The measurements revealed that the control traffic bandwidth between two controllers depends linearly on the number of switches and edges in the clusters and on the number of edges between the clusters. Using these results, we can build up a general model for networks with C controllers, and the control traffic generated by any controller i to any other controller can be expressed as

$$\begin{aligned} B_i = b^0 + b^s S_i + b^l L_i + b^{s-} \sum_{j \neq i} S_j + b^{l-} \sum_{j \neq i} L_j \\ + b^d \sum_{j \neq i} L_{ij} + b^e \sum_{j \neq i} \sum_{k \neq i,j} L_{jk}, \end{aligned} \tag{1}$$

where S_i is the number of switches in cluster i, L_i is the number of internal links in the cluster, and L_{jk} is the number of links where the end nodes belong to cluster j and k respectively. Parameter b^0 is the *zero bandwidth*, representing the control traffic that is generated even when the network does not contain any switches. Parameters b^s and b^l represent control information about switches and links in the own cluster, b^{s-} and b^{l-} about switches and links within other clusters, b^d about links with one end in the own cluster and b^e on links running among other clusters.

The measurement results in [4] show that the b^* (the * may denote s, l, $s-$, $l-$, d or e) parameters depend on the number of controllers C in the network. Due to the anti-entropy protocol, where the controller to communicate to is selected randomly, we could expect that $b^* \propto 1/C$, but the measured values for $C = 2$ and 3 show that b^* includes also a fix part. Therefore, we consider

$$b^* = b^*_f + \frac{1}{C-1} b^*_p, \tag{2}$$

where b^*_f is the fix part, and b^*_p contributes to the part that scales down inversely proportionally with C.

Table I shows the parameter values estimated from the measurements results in [4]. Note that from (2) we have $b^d_f = 0$, therefore we estimate $b^e_f \approx 0$. From the table we can also conclude that the dominating parameter is the zero bandwidth b^0 for small networks. For large networks the switch and link related control traffic has similar contribution.

TABLE I
BANDWIDTH PARAMETER VALUES. THE UNIT IS KBPS, THE $C = 2$ AND $C = 3$ VALUES ARE FROM [4]

Parameter	$C = 2$	$C = 3$	b^*_f	b^*_p
b^0	53	43	33	20
b^s	1.45	1.12	0.79	0.66
b^l	1.45	0.93	0.41	1.04
b^{s-}	0.48	0.35	0.22	0.26
b^{l-}	0.94	0.53	0.12	0.82
b^d	1.55	0.87	*0*	*1.6*
b^e	-	0.7	*0*	*0.7*

Finally, we derive the aggregated generated c2c control traffic, including all traffic generated by all the controllers to all the other controllers, resulting

$$\begin{aligned} B_{c2c} &= \sum_{i=1}^{C} (C-1) B_i \\ &= C(b^0_f (C-1) + b^0_p) \\ &\quad + S(b^{s-}_f (C-1)^2 + b^{s-}_p (C-1) + b^s_f (C-1) + b^s_p) \\ &\quad + L(b^{l-}_f (C-1)^2 + b^{l-}_p (C-1) + b^l_f (C-1) + b^l_p) \\ &\quad + L_d (2b^d + (C-2) b^e), \end{aligned} \tag{3}$$

where L is the number of intra-cluster links, and L_d is the number of inter-cluster links. Note that (3) provides a closed form expression of the control traffic that does not depend on the characteristics of the single clusters. We can conclude that the control traffic depends on the total number of controllers C, switches S, intra-cluster links L, and inter-cluster links L_d. As we see, B depends linearly on S, L and L_d, but increases quadratically with C, which introduces significant penalty as the number of controllers is increased.

The controllers also communicate with the switches in their own cluster, e.g., through the OpenFlow protocol, where switches query the controllers for routing information. We express the aggregated switch to controller traffic as

$$B_{s2c} = (S - C) 2 \lambda G, \tag{4}$$

where λ is the long term average of the number of messages generated by a switch per second, G is the average message size in bits. The expression reflects that the C controllers are hosted by C switches, and these switches do not generate s2c control traffic on the network links.

To proceed and derive the actual volume of control traffic in the network, we need to consider the length of the paths the control traffic is forwarded on. Therefore, from this point we consider specific topologies.

C. Network control traffic in regular grid topology

First we consider a wrapped two dimensional grid topology. We present detailed results for the rectangular grid, where each node has $k = 4$ neighbors, as shown on Figure 2. Then we generalize the results for some specific $k > 4$ values to see how k affects the control load.

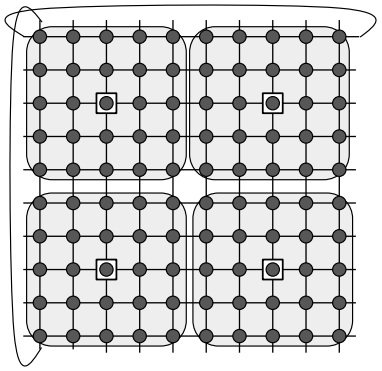


Fig. 2. Network forming a regular, folded grid with $k = 4$. Controllers (marked by squares) are placed regularly. Switches join the closest controller, which results equal size clusters (marked by shaded areas).

The S switches form C rectangular clusters of size $S_C = S/C$, and the controller is placed in one of the central switches to minimize the s2c control traffic as well as the maximum s2c delay. In the regular grid topology all controllers have statistically the same location, and consequently, all the B_i values are identical. Let P_{ij} be the length of the transmission path from controller i to controller j, and $\bar{P}^G_{c2c}$ the average controller to controller path length. We can then express the aggregate controller to controller load as

$$\begin{aligned} T^G_{c2c} &= \sum_{i=1}^{C} \sum_{j=1, j\neq i}^{C} B_i P_{ij} \\ &= B_i \sum_{i=1}^{C} \sum_{j=1, j\neq i}^{C} P_{ij} \\ &= B_i C(C-1) \frac{\sum_{i=1}^{C} \sum_{j=1, j\neq i}^{C} P_{ij}}{C(C-1)} \\ &= C(C-1) B_i \bar{P}^G_{c2c} \\ &= B^G_{c2c} \bar{P}^G_{c2c}. \end{aligned} \tag{5}$$

To express B^G_{c2c}, we need to replace the topology dependent parameters in (3), that is, the number of inter-cluster links and the number of intra-cluster links with the grid topology specific values. In the case of $k = 4$, the number of inter-cluster links leaving one cluster is equal to the length of the cluster border $4\sqrt{S_C}$, thus, for the entire network we get

$$L^G_d = C \frac{4\sqrt{S_C}}{2} = 2\sqrt{SC}, \tag{6}$$

then, sice altogether there are $2S$ links, the number of intra cluster links can be calculated as

$$L^G = 2S - 2\sqrt{SC}. \tag{7}$$

Finally, we need to derive the average path length $\bar{P}^G_{c2c}$. We notice that in the considered topology, shortest paths connecting far away controllers can be constructed from path segments between neighboring controllers. Therefore

$$\bar{P}^G_{c2c} = \bar{H}^G_{c2c} \sqrt{\frac{S}{C}}, \tag{8}$$

where $\sqrt{S/C}$ is the distance of neighboring controllers and $\bar{H}^G_{c2c}$ is the average number of path segments to travel from any controller to any controller.

Let us derive $\bar{H}^G_{c2c}$. We restrict the derivation for odd $\sqrt{C}$ values. We can see that each controller needs to reach other controllers in maximum $(\sqrt{C}-1)/2$ steps away in both horizontal and vertical directions. Therefore, for odd $\sqrt{C}$ we get the average number of path segments to travel

$$\bar{H}^G_{c2c} = \frac{\sqrt{C}}{2}, \tag{9}$$

which provides

$$\bar{P}^G_{c2c} = \frac{\sqrt{C}}{2} \sqrt{\frac{S}{C}} = \frac{\sqrt{S}}{2}. \tag{10}$$

$\bar{H}_{c2c}$ is somewhat higher for even $\sqrt{C}$ values, but the difference diminishes as C increases, and therefore we use (10) in the followings.

Substituting (3) and (10) into (5), we get

$$\begin{aligned} T^G_{c2c} = \frac{S^{3/2}}{2} \Bigg[& \frac{1}{S_C} \Big(b^0_f \Big(\frac{S}{S_C} - 1 \Big) + b^0_p \Big) \\ & + \Big(b^{s-}_f \Big(\frac{S}{S_C} - 1 \Big)^2 + (b^{s-}_p + b^s_f) \Big(\frac{S}{S_C} - 1 \Big) + b^s_p \Big) \\ & + 2 \Big(1 - \frac{1}{\sqrt{S_C}} \Big) \Big(b^{l-}_f \Big(\frac{S}{S_C} - 1 \Big)^2 \\ & \qquad + (b^{l-}_p + b^l_f) \Big(\frac{S}{S_C} - 1 \Big) + b^l_p \Big) \\ & + \frac{2}{\sqrt{S_C}} \Big(2b^d + \Big(\frac{S}{S_C} - 2 \Big) b^e \Big) \Bigg], \end{aligned} \tag{11}$$

where $S_C = S/C$ is the size of the clusters. From (11) we can conclude that the controller to controller traffic increases with $S^{7/2}$, while it decreases inversely proportionally with the cluster size. This would motivate the use of few controllers, to decrease the controller to controller traffic.

Let us now consider the control traffic generated by the switch to controller messages, given by (4). The optimal position of the controller is in the middle of a cluster. Then the average length of the switch to controller transmission path can be calculated similarly to $\bar{H}^G_{c2c}$, but considering the grid of switches, instead of the grid of controllers, which gives $\bar{P}^G_{s2c} = \sqrt{S_C}/2$, and the total switch to controller traffic load becomes

$$T^G_{s2c} = B^G_{s2c} \bar{P}^G_{s2c} = \lambda G S \sqrt{S_C} \Big(1 - \frac{1}{S_C} \Big). \tag{12}$$

From (11) and (12) we see that T_{c2c} decreases, while T_{s2c} increases with the cluster size S_C. Therefore, as expected, there is a cluster size that minimizes the control traffic for

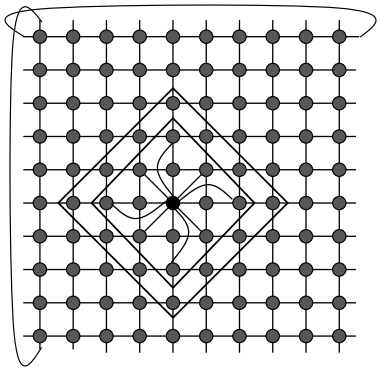


Fig. 3. Network forming a regular, folded grid. Neighborhood areas for $k = 12$ and 24, resulting in $l = 2$ and 3 respectively are shown. For the $k = 12$ case we also show the connection pattern.

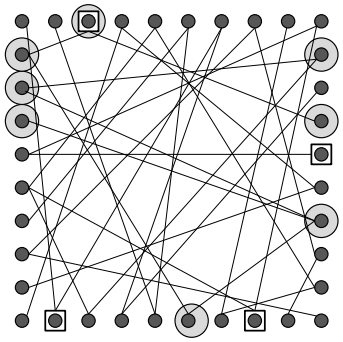


Fig. 4. Network forming an Erdős-Rényi random graph. Controllers (marked by squares) are placed randomly. Switches join the closest controller, which results clusters of variable size. Some nodes of the same cluster are marked by shaded circles.

given network size and the intensity of the switch to controller traffic λ.

To generalize the results, we consider grid topologies where each node can reach its k closest neighbors, resulting in a denser, but still regular grid. To simplify the derivations, while still addressing large networks, we consider the specific k values, where it holds that shortest c2c paths can be constructed using only horizontal and vertical edges. E.g., this is the case for $k = 12$ and 24, where single hop neighbors form a rectangular as shown on Figure 3. Also we assume that the cluster diameters are much larger than the longest single hop in the grid, and thus the derivation steps we followed for $k = 4$ are still valid. That is, we need to derive the k specific L^G, L^G_d, $\bar{P}^G_{c2c}$ and $\bar{P}^G_{s2c}$ expressions.

Let us denote by l the length of the longest transmission, measured in the distance of the neighboring nodes (e.g., for $k = 4$ $l = 1$, and for $k = 12$ shown on Figure 3, $l = 2$. Inspecting Figure 3 we can derive the relationship between k and l

$$k = 4(1 + 2 + ... + l) = 2l(l + 1), \quad l = \frac{\sqrt{1 + 2k} - 1}{2}. \tag{13}$$

We start by deriving L^G_d. Now all the nodes that are within distance l of the cluster border have inter-cluster links, giving

$$L^G_d = 2\sqrt{C}\sqrt{S}\sum_{i=0}^{l-1}(l - i)(2i + 1) = \ldots = \frac{\sqrt{SC}k\sqrt{1 + 2k}}{6},$$

$$L^G = \frac{1}{2}kS - L^G_d. \tag{14}$$

Equation (9) still holds, while the distance of neighboring controllers becomes $\sqrt{S/C}/l$. This provides the average controller to controller transmission path

$$\bar{P}^G_{c2c} = \frac{\sqrt{S}}{2l} = \frac{\sqrt{S}}{\sqrt{1 + 2k} - 1}. \tag{15}$$

We can see k has opposite effects on L^G, L^G_d and $\bar{P}^G_{c2c}$. The final T^G_{c2c} expression becomes

$$\begin{aligned} T^G_{c2c} = \frac{S^{3/2}}{\sqrt{1 + 2k} - 1}\Bigg[& \frac{1}{S_C}\left(b^0_f\Big(\frac{S}{S_C} - 1\Big) + b^0_p\right) \\ & + \left(b^{s-}_f\Big(\frac{S}{S_C} - 1\Big)^2 + (b^{s-}_p + b^s_f)\Big(\frac{S}{S_C} - 1\Big) + b^s_p\right) \\ & + \frac{k}{2}\Big(1 - \frac{\sqrt{1 + 2k}}{3\sqrt{S_C}}\Big)\left(b^{l-}_f\Big(\frac{S}{S_C} - 1\Big)^2\right. \\ & \qquad \left. + (b^{l-}_p + b^l_f)\Big(\frac{S}{S_C} - 1\Big) + b^l_p\right) \\ & + \frac{k\sqrt{1 + 2k}}{6\sqrt{S_C}}\left(2b^d + \Big(\frac{S}{S_C} - 2\Big)b^e\right)\Bigg]. \end{aligned} \tag{16}$$

The c2c traffic still increases with $S^{7/2}$. Larger part of the traffic decreases with $\sqrt{k}$, due to the longer single hop transmissions, however, the traffic has a component that increases linearly with k, due to the increasing number of edges that has to be reported in the ONOS messages.

Considering the switch to controller traffic, the longer links lead to shorter transmission paths, giving

$$\bar{P}^G_{s2c} = \frac{\sqrt{S_C}}{2l} = \frac{\sqrt{S_C}}{\sqrt{1 + 2k} - 1}. \tag{17}$$

This makes T^G_{s2c} decrease with increasing $\sqrt{k}$, that is, the increased network density effects only positively the switch to controller traffic.

D. Network control traffic under random topology

Let us now consider a network with random topology, specifically, a network of S switches, and links forming an Erdős-Rényi (ER) random graph, where two switches are connected with probability p. Since all switches see statistically similar neighborhood, we place the C controllers at random switch locations, and we assume, that each of the switches connects to the closest controller. A possible realization of such a network is shown on Figure 4.

The ER graph is homogeneous in the sense that from each node the path length distribution to all other nodes can be characterized by the same CDF $F(x)$ and pdf $f(x)$. Then, the probability that a given switch i with path length x_i selects controller j as its closest controller is the probability that the paths to the other controllers are longer than x_i, that is $\bar{F}(x_i)^{C-1}$. The probability that an arbitrary node selects controller j becomes $p_j = \int \bar{F}(x)^{C-1} f(x) dx$. Note, that the right hand side of this equation is independent from j, and therefore has to be the same for each controller, that is, $p_j = p = 1/C$. Consequently, the size of the clusters is binomially distributed with parameters S and $1/C$, with an average cluster size of S/C.

We follow (5) to calculate the total controller to controller traffic load. Since now we have random topology, we express the mean value, $\bar{T}^{ER}_{c2c} = \sum_{i=1}^{C} \sum_{j=1, j\neq i}^{C} B_i P_{ij} = \bar{B}_{c2c} \bar{P}_{c2c}$.

Again we need the topology specific values of L and L_d to express $\bar{B}_{c2c}$. Since now we have a random topology, the L and L_d values are random numbers. As B_{c2c} depends linearly on L and L_d, we use the average values to express $\bar{B}_{c2c}$. Let us introduce $\bar{k} = (S-1)p$ as the average number of links a switch has. The expected number of intra-cluster links of a switch s is

$$\bar{L}_s = E[L_s] = \sum_{q=1}^{S} \sum_{n=0}^{S-1} E[L|q,n] = \frac{(\frac{S}{C}-1)\bar{k}}{S-1}, \tag{18}$$

where n denotes the number of links the switch has, and q the size of the cluster it belongs to, and both of these random variables have Binomial distribution $B(S-1,p)$ respectively $B(S,1/C)$. $E[L|q,n]$ is the expected number of intra-cluster links under given n and q values, with distribution $B(n,q/S)$.

Then, for the average number of intra-cluster respectively inter-cluster edges in the entire network we get

$$\bar{L} = \frac{S}{2}\bar{L}_s = \frac{S(S_C-1)\bar{k}}{2(S-1)}, \tag{19}$$

$$\bar{L}_d = \frac{S\bar{k}}{2} - \bar{L} = \frac{SS_C(C-1)\bar{k}}{2(S-1)}, \tag{20}$$

where $S_C = S/C$ as before.

Now let us consider the average length of the transmission paths between the controllers. We use the results of [21], that gives the CCDF of the path length between any two nodes in an ER graph as

$$\bar{F}(x) = \exp\Big[\frac{\bar{k}^x}{S-1}\Big], \tag{21}$$

and the average path length, which is also the average c2c path length as

$$\bar{P}^{ER}_{c2c} = \frac{\ln(S)-\gamma}{\ln \bar{k}} + \frac{1}{2}, \tag{22}$$

where $\gamma = 0.5572$ is the Euler constant.

Comparing these parameters to the ones in the regular grid, we see that now a larger ratio of links are inter-cluster links, which slightly increases the generated control traffic according to Table I. Most importantly, however, the average path lengths have very different characteristics for the two topologies. In the grid topology it increases with $\sqrt{S}$, while in the random graph only with $\ln S$, which shows already that the total controller to controller traffic load will be lower under the random graph topology. We see as well that the number of neighbors $\bar{k}$ has stronger effect in the grid topology.

The final expression of $\bar{T}^{ER}_{c2c}$ is

$$\begin{aligned}\bar{T}^{ER}_{c2c} = \Big(\frac{\ln(S)-\gamma}{\ln \bar{k}} + \frac{1}{2}\Big)\Big(\frac{S}{S_C}\Big(b^0_f\Big(\frac{S}{S_C}-1\Big)+b^0_s\Big) \\ + S\Big(b^s_{f-}\Big(\frac{S}{S_C}-1\Big)^2 + (b^s_{s-}+b^s_f)\Big(\frac{S}{S_C}-1\Big)+b^s_s\Big) \\ + \Big(\frac{(S_C-1)\bar{k}}{2}\Big)\Big(b^l_{f-}\Big(\frac{S}{S_C}-1\Big)^2 + (b^l_{s-}+b^l_f)\Big(\frac{S}{S_C} \\ -1\Big)+b^l_s\Big) + \frac{(S-S_C)\bar{k}}{2}\Big(2b^d + \Big(\frac{S}{S_C}-2\Big)b^e\Big)\Big).\end{aligned} \tag{23}$$

As expected, the increase of the traffic in the number of switches S is lower than in the case of the regular grid, but still the traffic scales with $S^3 \ln(S)$.

Finally, let us consider the switch to controller control traffic load, which depends on the average switch to controller path length. Let X_i denote the random variable representing the path length from a node to controller i, with CCDF given in (21), and let $Y = \min(X_1, X_2, ...X_c)$ be the path length to the closest controller. Then

$$\bar{F}_Y(x) = (\bar{F}(x))^C = \exp\Big[\frac{C\bar{k}^x}{S-1}\Big], \tag{24}$$

and following [21] we can estimate the mean value of Y, $\bar{P}^{ER}_{s2c}$ as

$$\bar{P}^{ER}_{s2c} = \frac{\ln(\frac{S}{C})-\gamma}{\ln \bar{k}} + \frac{1}{2} = \frac{\ln(S_C)-\gamma}{\ln \bar{k}} + \frac{1}{2}. \tag{25}$$

Since P_{s2c} is proportional to $\ln(S_C)$, the increase in cluster size does not increase the path length significantly. This is a good phenomena as shorter path length guarantees low latency and less traffic in the network. We also see that the positive effect of increasing $\bar{k}$ will be small at large $\bar{k}$ values, and $\bar{k}$ does not change the characteristics of the scaling in network size.

Combining (4) and (25) we can write total switch to controller traffic in the network as

$$\bar{T}^{ER}_{s2c} = \lambda G\Big(1-\frac{1}{S_C}\Big)\Big(\frac{2(\ln(S_C)-\gamma)}{\ln \bar{k}} + 1\Big), \tag{26}$$

that is, overall the switch to controller traffic increases with the cluster size, however, this increase is only logarithmic. Similarly to the regular grid, the traffic is proportional to the intensity and the size of the switch to controller queries.

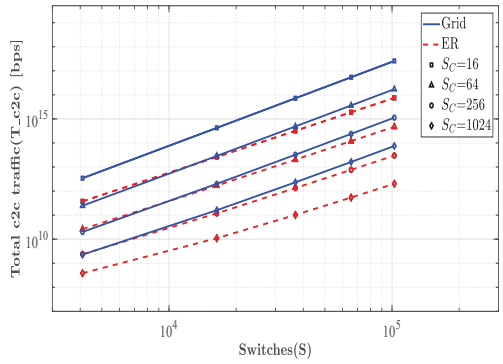


Fig. 5. Controller to controller traffic in regular grid and ER networks as a function of the number of switches S, and for different clusters sizes S_C ($\bar{k} = 4$).

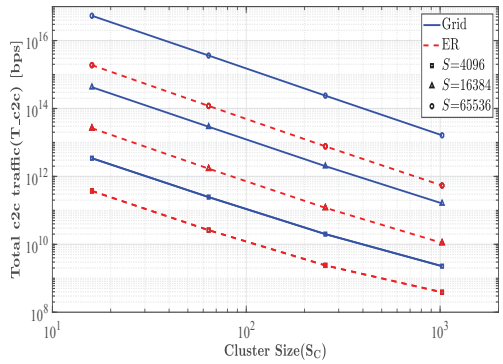


Fig. 6. Controller to controller traffic in regular grid and ER networks, as a function of the cluster size S_C, and for different number of switches S ($\bar{k} = 4$).

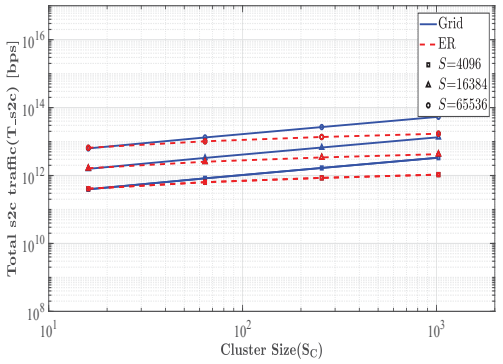


Fig. 7. Switch to controller traffic in regular grid and ER networks as a function of the cluster size S_C and for different number of switches S ($\bar{k} = 4$, $G = 128$Bytes and $\lambda = 25$kpps).

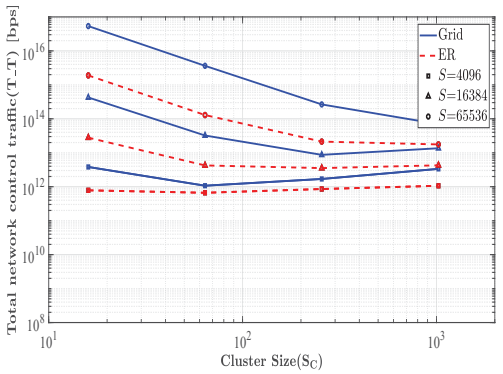


Fig. 8. Total control traffic in regular grid and ER networks as a function of the number of switches S, and for different cluster sizes S_C ($\bar{k} = 4$, $G = 128$Bytes and $\lambda = 25$kpps). Optimal cluster sizes are marked.

V. Numerical examples

In this section we use the expressions derived in Section IV to evaluate the effect of the network size, the cluster size, the intensity of the switch to controller traffic, and the network topology on the value of the control traffic emerging in the network, and discuss the selection of the preferable number of controllers. To be able to evaluate scalability, we consider network sizes in the range of $10^5 - 10^5$ switches, which corresponds to networks of a large number of autonomous systems. Unless otherwise noted, we consider $\bar{k} = 4$, s2c control packet size of $G = 128$Bytes and per switch packet generation rate $\lambda = 25$kpps [14]. The parameter values of the ONOS control traffic are given in Table I.

Figure 5 shows total $c2c$ traffic in the network as a function of the number of switches, for a given cluster sizes for grid and ER networks, on a log-log plot. It reflects well the analytic results on the scaling of the traffic in S. We note that the gradient, giving the exponent of S, is only slightly lower in the ER graph, however, the absolute values differ in ca. one order of magnitude. We see as well that the cluster size has significant effect. The traffic level is very high in general, in the order of terabits per second.

Let us next evaluate the amount of the controller to controller traffic as a function of the cluster size. Figure 6 shows that the decrease of the control traffic is similar in the grid and ER topologies, and is dominated by the parts of (11) and (23) with quadratic decrease, which would motivate the use of large clusters in both topologies.

In contrast, we expect that decreasing cluster size increases the switch to controller traffic. As we see on Figure 7, this increase is significant in the case of the grid topology where the switch to controller path length is proportional to $\sqrt{S_C}$, while less dominant in the ER case, where the path length to the closest controller increases with $\ln S_C$.

Figure 8 gives the total traffic and optimum cluster size for both topologies. As the c2c traffic is dominating in large networks, the optimal cluster size is also large. We see that in middle-sized networks the traffic decreases significantly at smaller than optimal cluster sizes, while there is little penalty if the cluster size is too large. In small networks, under and over dimensioned cluster sizes have similar effect. In general, the cluster size that minimizes the total traffic is smaller

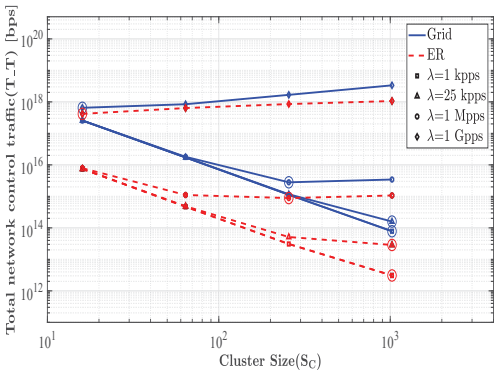


Fig. 9. Total control traffic in regular grid and ER networks as a function of the cluster size S_C, and for different switch to controller traffic intensity values λ (S = 16384, $\bar{k} = 4$ and $G = 128$Bytes). Optimal cluster sizes are marked.

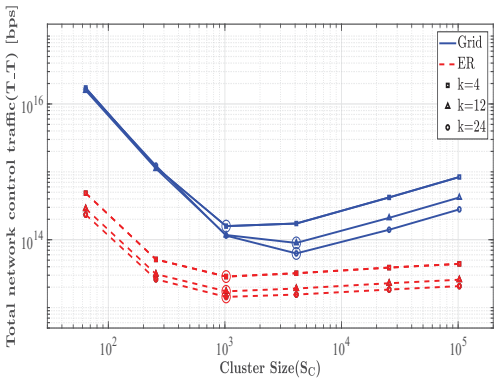


Fig. 10. Total control traffic in regular grid and ER networks as a function of the cluster size S_C, but for different degree values $\bar{k}$ (S = 102400, $G =$ 128Bytes and $\lambda = 25$kpps)). Optimal cluster sizes are marked.

in smaller networks, and, for the considered parameters, the optimal values are similar for the two network topologies.

Figure 9 compares the total traffic for different switch to controller query rates λ and fixed network size $S = 16384$. The figure shows that the intensity of the s2c traffic has significant effect not only on the traffic levels, but also on the preferred cluster size. High λ allows only smaller clusters, while under low traffic intensity larger clusters can decrease the control traffic significantly. Therefore the network should be configured, or even reconfigured according to the s2c traffic.

Finally, on Figure 10 we evaluate the effect of the network density, that is, the average node degree $\bar{k}$, considering k values from 4 up to 24. The analytic results already showed us that $\bar{k}$ does not affect how the traffic volume depends on the network size S and on the cluster size S_C, but scale the volume of the traffic. Increased network density has the negative effect of increasing the amount of information that needs to be exchanged in the control traffic, but it also have the positive effect of shorter transmission paths. In the grid network, these two are in balance when the c2c traffic dominates, and we see the positive effect of increased $\bar{k}$ only when the s2c traffic becomes significant. In the ER network the positive effect is visible for all cluster sizes, but the gain diminishes already at around $\bar{k} = 24$, even for the considered large network.

VI. Conclusions

In this paper we have modelled the volume of the control traffic in distributed SDNs, when ONOS is used for maintaining the shared view of the network. We evaluated the effect of the network size, the number of controllers in the network, and the network topology. We have seen that regular grid networks will have high control traffic due to the long transmission paths, but even random graphs with small-world property suffer from heavy control traffic. We have demonstrated that there is an optimal cluster size that minimizes the volume of the control traffic, and that this cluster size depends significantly on the intensity of the switch to controller traffic, but is often less sensitive to the network topology itself. Similar studies could also help the network design in specific application areas, like data center networking [22]

We have shown that the density of the network have contradicting effects on the control traffic, and little is gained in well-connected networks. Our results consider full mesh connectivity among the controllers. This ensures the fast convergence of the anti-entropy algorithm, but leads to heavy control traffic, due to the long controller to controller paths. A further optimization of the ONOS protocol thus could employ different connection patterns, to trade off consistency and control traffic load.

References

[1] B. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, March 2014.

[2] M. Caria, A. Jukan, and M. Hoffmann, "A performance study of network migration to sdn-enabled traffic engineering," in *Proc. of IEEE Globecom*, 2013.

[3] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," in *Proc. of the NEM Summit*, 2012.

[4] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *Proc. of IEEE ICC*, May 2016.

[5] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, March 2015.

[6] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, Aug 2014.

[7] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 514–527, Sept 2017.

[8] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 19, no. 1, pp. 30–33, Jan 2015.

[9] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient SDN architecture," in *Proc. of IEEE Design of Reliable Communication Networks (DRCN)*, 2016.

[10] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Computer Communications*, vol. 77, pp. 41 – 51, March 2016.

[11] T. Zhang, P. Giaccone, A. Bianco, and S. D. Domenico, "The role of the inter-controller consensus in the placement of distributed SDN controllers," *Computer Communications*, vol. 113, pp. 1 – 13, Sept 2017.

[12] P. Megyesi, A. Botta, G. Aceto, A. Pescapè, and S. Molnár, "Available bandwidth measurement in software defined networks," in *Proc. of ACM Symposium on Applied Computing*, 2016.

[13] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the sdn control traffic in large isp networks," in *2015 IEEE International Conference on Communications (ICC)*, June 2015.

[14] D. Levin, A. Wundsam, A. Feldmann, S. Seethamaran, M. Kobayashi, and G. Parulkar, "A first look at OpenFlow control plane behavior from a test deployment," Technische Universitt Berlin, Fakultt Elektrotechnik und Informatik, Tech. Rep., 2011.

[15] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, and W. Snow, "ONOS: towards an open, distributed SDN OS," in *Proc. of ACM Worshop on Hot Topics in Software Defined Networking*. ACM, 2014.

[16] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *Proc. of IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014.

[17] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "CAP for networks," in *Proc. of ACM SIGCOMM workshop on Hot Topics in Software Defined Networking*, 2013.

[18] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. of USENIX Annual Technical Conference*, 2014.

[19] R. A. Golding and D. D. Long, "Simulation modeling of weak-consistency protocols," *Network Systems Design*, Jan 1999.

[20] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[21] J. A. H. Agata Fronczak, Piotr Fronczak, "Average path length in random networks," *Physical Review E, vol. 70, Issue 5, id. 056110*, pp. 1–7, 2004.

[22] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (dcn): Infrastructure and operations," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 640–656, 2017.

HARMLESS: Cost-Effective Transitioning to SDN for Small Enterprises

Levente Csikor*, László Toka†‡, Márk Szalay†, Gergely Pongrácz§, Dimitrios P. Pezaros*
and Gábor Rétvári†
*School of Computing Science, University of Glasgow, Email: {firstname.lastname}@glasgow.ac.uk
†High Speed Networks Lab, Budapest University of Technology and Economics, Email: {lastname}@tmit.bme.hu
‡MTA-BME Network Softwarization and Information Systems Research Groups
§Ericsson Research, TrafficLab Hungary, Email: gergely.pongracz@ericsson.com

***Abstract*—Software-Defined Networking (SDN) offers a new way to operate, manage, and deploy communication networks and to overcome many long-standing problems of legacy networking. However, widespread SDN adoption has not occurred yet due to the lack of a viable incremental deployment path and the relatively immature present state of SDN-capable devices on the market. While continuously evolving software switches may alleviate the operational issues of commercial hardware-based SDN offerings, namely lagging standards-compliance, performance regressions, and poor scaling, they fail to match the cost-efficiency and port density.**

In this paper we propose HARMLESS, a new SDN switch design that seamlessly adds SDN capability to legacy network gear, by emulating the OpenFlow switch OS in a separate software switch component. This way, HARMLESS enables a quick and easy leap into SDN, combining the rapid innovation and upgrade cycles of software switches with the port density and cost-efficiency of hardware-based appliances into a fully dataplane-transparent and vendor-neutral solution. HARMLESS incurs an order of magnitude smaller initial expenditure for an SDN deployment than existing turnkey vendor SDN solutions while it yields matching, or even better data plane performance for smaller enterprises.

***Index Terms*—SDN, Migration, OpenFlow, Switch design**

I. Introduction

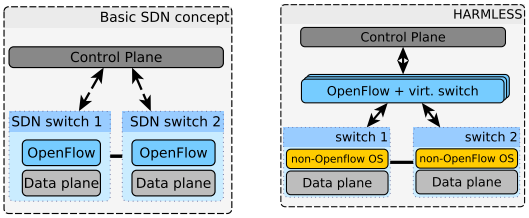
SDN offers a radical break with traditional ways of building, operating and managing networks. By the physical and logical separation of the network control plane from the packet processing functionality, SDN exposes new levels of *abstraction* to the operator. SDN hides the specifics of the underlying data plane technologies from the network control applications behind a standardized southbound interface, and unprecedented network and service *programmability*, since the network is now controlled by an adaptable software functionality via an open API. Migration to SDN architecture improves network operations by eliminating the need for box-by-box management and troubleshooting, eases to create network functions and services due to the flexibility of the global network view in the centralized SDN control plane, and allows the operator to easily buy into new models for network management and operations, like automated orchestration, on-the-fly chaining of services, and "as-a-service" schemes [1]–[4].

ISBN 978-3-903176-08-9 © 2018 IFIP

Despite the fact that many large corporations (e.g., Google [5]) and telcos (e.g., Deutsche Telekom [6]) have already gained significant foothold in SDN, *smaller businesses, campus network operators, and service providers* without substantial in-house expertise and select IT staff ("organizations that aren't called Google" [7]) face significant business, economic, and technical deployment barriers, since migration to SDN requires a nontrivial amount of forward planning, an extensive investigation of vendor offerings/options, and a fairly radical change in the mental model, producing a typical chicken and egg problem [1]–[4], [7].

First, there is a broad selection of SDN migration strategies: incremental deployment strategies may offer the smoothest upgrade path and the least interference with daily network operations [8], yet managing heterogeneous network architectures may prove challenging [2], [4]. Jumping outright to full-blown SDN by swapping all legacy network gear to SDN-capable devices overnight may mitigate this pain factor, but greenfield migration is hardly an option for small businesses due to the huge capital expenditure, the flag-day deployment, and the induced service downtime. *A lightweight SDN migration combining the smoothness and reversibility of incremental upgrades with the swiftness and transparency typical to greenfield deployments is still largely missing* [9].

Second, there is the paradox of choice inherent to the booming and immature state of the SDN market today, with a breadth of vendor SDN offerings, turnkey solutions, and marketing hype, that a less informed operator may find very challenging to explore and evaluate [1], [6]. For a starter, to obtain an SDN-enabled appliance a network operator has essentially two nontrivial choices: buying commercial off-the-shelf (COTS) and white-box (i.e., generically branded switches with no default network operating system) hardware switches or relying on, possibly already purchased and deployed, general-purpose servers and running a virtual software switch on top (e.g., Open vSwitch, OVS [10]). Thanks to the use of special-purpose ASICs and network processors, hardware SDN network gear has traditionally been praised for providing high port density at a reasonable price tag, albeit notorious for lacking standards-compliance, limited TCAM sizes (typical 1^{st} and 2^{nd} generation devices can have at most 100s and couple of 1000s of flow rules in TCAM), performance



(a) SDN with OpenFlow (b) HARMLESS

Fig. 1: HARMLESS: SDN with an extra level of separation.

regressions, and unscalability [9], [11], [12]. In particular, users are widely complaining about missing OF features (even as basic as IP address rewrite [9]), switch control plane performance and delays in updating the data plane, atomic flow modification commands not being applied atomically or at all [11], [13], etc. Note that some newer generation OF switches (e.g., `NoviFlow, Corsa, Barefoot`) offering high performance with up to 1 million flow rules have recently become available in the market, however not just their costs hinders smaller enterprises to obtain one, but due to the way ASICs are designed an arbitrary forwarding pipeline cannot be applied without fulfilling certain requirements [14] (e.g., wire-speed VLAN handling can only be done in table 0).

Software SDN switches, on the other hand, struggle to match the port density of hardware switches due to the physical space limits of blades and the steep price of multi-port NICs, but at the same time excel at programmability, extensibility, rapidly evolve with new standards and receive bug fixes very fast [10], [15]. While, thanks to recent advances in softswitch design and implementation (e.g., multi-threaded switch design, hierarchical flow caching, custom-compiled OF datapaths [10], [15]–[19]) the performance tax of software switching has greatly decreased, programmability and port density are still competing, if not mutually exclusive, goals in current SDN networking equipment.

In this paper, we propose HARMLESS, the *Hybrid ARchitecture to Migrate Legacy Ethernet Switches to SDN*, to foster SDN migration for smaller enterprises. HARMLESS leverages the current trend for "software-defined-everything", but takes this idea to the extreme: it applies an additional level of abstraction on top of the conventional control plane–data plane separation by *further decoupling the packet processing hardware from the switch's operating system*, which are today integrated in COTS devices in a single box, *and implementing the OpenFlow (OF) OS in a dedicated software switch* (see Fig. 1). This makes it possible to add SDN capability to plain Ethernet switches, or to any legacy network device for that matter, through bypassing the legacy switch OS. Thanks to the additional level of virtualization, *HARMLESS realizes a delicate sweet spot between hardware and software SDN switching*. In particular, *it combines the advantages of software and hardware switching, whereas the hardware component delivers the high port density and raw packet processing functionality, and the softswitch adds programmability, adaptability, and standards-compliance*. Using extensive measurements with a HARMLESS prototype, we show that the benefits of HARMLESS are realized with no significant impact on raw packet processing performance, latency, and dataplane transparency (note that the performance is upper bounded by the used software switch).

From an economical aspect, *HARMLESS offers a viable migration strategy to smaller enterprises.* Since HARMLESS leverages the *existing* network infrastructure it offers distinct price advantages over SDN alternatives available on the market (see details in Sec. III). Crucially, in cases where legacy switches and bare-metal servers for running the OF component are readily available, like in smaller private clouds, enterprise and research networks, *HARMLESS makes it possible to get into SDN instantly, incurring zero expenditure for a partial or even a complete deployment.* And even if equipment must be purchased anew, HARMLESS can save up to an order of magnitude investment. In a broader perspective, HARMLESS sheds a fresh new light on the ages-old, and often highly contentious, "hard switch vs softswitch" debate and presents an interesting new dimension in switch architectures [20]–[25].

Roadmap: in Sec. II we present the HARMLESS architecture, in Sec. III we give a cost analysis, in Sec. IV we evaluate HARMLESS in many aspects, in Sec. V we summarize related work, and finally Sec. VI concludes the paper.

II. The HARMLESS Architecture

So how to magically turn a legacy device, say, a dumb Ethernet switch, into an OpenFlow-speaking one? After all, this would require to open up what is traditionally a closed black box and substitute the legacy switch OS with an SDN-capable one, something that has proved notoriously difficult to do so far. Instead, we adopt a more viable and backward compatible approach for the purposes of HARMLESS by extending the "Tagging and Hairpinning" technique (also called "anything-on-a-stick" [26], [27] or distributed switch design [28]), originally advocated for hypervisor switches by Cisco and HP, to the general context of SDN [24].

The idea behind "Tagging and Hairpinning" is to offload VM-to-VM communication from the hypervisor to the first hop switch. When a VM sends a packet it is marked by a unique VLAN id ("tagging") and forwarded to the access switch, which will then do a forwarding/policy lookup to decide whether to loop the packet back to another VM, in which case it is marked with the unique VLAN id of the target VM ("hairpinning"), or send it further along the data center fabric, or drop it right away. The rationale for this technique is that packet processing is done on efficient special purpose hardware at the first hop switch instead of a potentially less powerful hypervisor switch, while the downsides are doubling bandwidth utilization and increased latency. The main contribution of this paper is the observation that, when cast in the general context of SDN switching, *the "Tagging and Hairpinning" technique yields a uniquely cost-efficient organization of packet processing functionality and forwarding intelligence, and presents an appealing incremental SDN deployment path.*

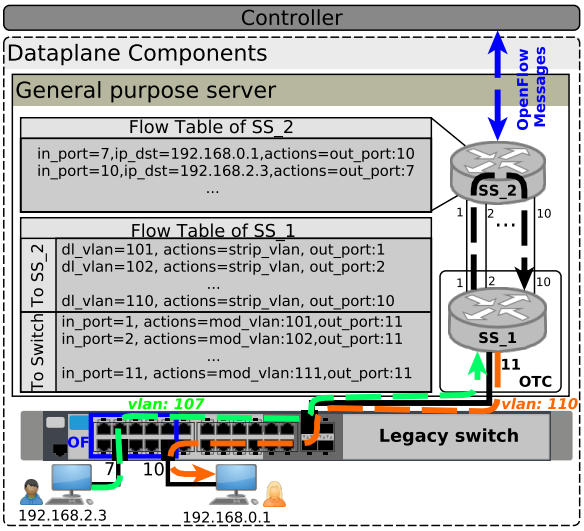


Fig. 2: Transparent HARMLESS: Software Switch Twice (S4).

For the purposes of presenting HARMLESS, suppose that an operator is given a manageable Layer-2 (L2) 802.1Q Ethernet switch with free high-speed trunk ports, a general purpose server that has spare NICs and available capacity to run an OpenFlow vswitch, and adequate cabling, backplane capacity, or other means for interconnecting the switches' trunk ports with the softswitch NICs (see bottom of Fig. 2). Suppose further that a host with IPv4 address `192.168.2.3` connected to port 7 on the legacy switch wishes to send packets to another host with address `192.168.0.1` connected to port 10 on the same switch, and suppose that a security policy is in place according to which these two hosts are permitted to exchange traffic only between each other. Accordingly, suppose that the operator wants to control the switch through OpenFlow and so wishes to program this forwarding behavior as given by the *Flow table of SS_2* in Fig. 2. This would handle communication between the two hosts adequately, except that it would be impossible to control the legacy switch through OpenFlow due to the black box nature of legacy COTS appliances.

This is where the "Tagging and Hairpinning" technique comes in: let the legacy switch tag each packet with a unique VLAN id that identifies the access port it was received from, forward the tagged packet to the software switch along the trunk-port–softswitch interconnect (the *uplink*) to enforce the OpenFlow forwarding and security policies, and send the packet back to the legacy switch via another uplink "hair-pinned", i.e., tagged with the unique VLAN id of the proper outgoing port as per the specified flow table. (If the packet was already VLAN-tagged, the legacy switch can use VLAN Q-in-Q tunneling [29] or any other tagging scheme.)

In a strawman's approach, the controller would need to program a slightly modified flow table into the software switch, whereas the "match on ingress port X" rules are converted to "match on VLAN id X" rules and the "output to port Y" actions are rewritten to "modify VLAN id to Y and output to default port" actions. However, to avoid having to tailor controller programs to the underlying HARMLESS layer, we introduce a transparent HARMLESS setup, where the novel idea is to carefully extend the framework with an additional *OpenFlow Translator Component (OTC)* to serve as an adaptation layer. OTC is realized by an additional softswitch [30], which is connected to the OF component by as many patch ports (10 in our example) as the number of the access ports managed and dispatches packets to and from patch ports based on the VLAN ids (see Fig. 2). In our particular example, upon receiving a packet on port 7 from the first host, the legacy switch would tag it with a unique VLAN id, say, 107, and send it along the uplink. The OTC then dispatches the packet (with the VLAN tag removed) to patch port 7 towards the the softswitch SS_2 (managed by the controller), which in turn identifies the original input port based on the patch port and makes sure that the originating host is allowed to communicate with the destination host by matching the first flow entry in the flow table (see Flow Table of SS_2 in Fig. 2). Then, SS_2 sends out the packet on its patch port 10 towards the OTC, which pushes a unique VLAN id, say, 110 onto the packet, then it is looped back to the legacy switch, which, after doing a VLAN-to-port translation, forwards it to the destination host (on physical port 10). Note that the only requirements for the legacy switch are manageability (to setup the VLANs on the access ports), support for 802.1Q (to do the VLAN un/tagging), and free trunk ports to be used for an uplink (later, in Sec. IV we will deal with the "loss" of trunks), which allows to use HARMLESS over basically any legacy Ethernet switch on the market today [31], [32].

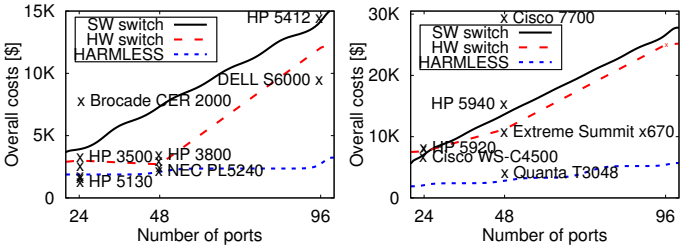
These modifications render HARMLESS data plane-transparent enabling to write controller programs ignoring the fact that the underlying data plane is realized with HARMLESS, to make controller programs portable between deployments, and to allow to invoke higher-level languages and policies to setup the data plane [33], [34].

III. Cost Analysis

Below, we argue that HARMLESS strikes a fine balance between the cost-efficiency of COTS switches and the flexibility of softswitches in terms of deployment costs.

A. Hard vs. Softswitch

The great majority of COTS and white-box SDN switch market options come with 24 or 48 ports at 1G (or 10G), supplemented with 2–4 uplink ports operating at 10G (or 40G, respectively) in 1U or 2U form-factor. In case of carrier grade port rates, e.g., 40G, the vendor offerings are even more diverse, often having no designated uplink ports. Prices vary in a wide range; the cheapest OpenFlow-capable 24x1G-port switch (HP-2920) can be purchased at $\$1,200$ while the most expensive 48x10G offering comes at a whopping $\$30,000$. The 48-port form factor seems to provide the most economic per-port prices, with the average price tag of $\$2,700$ for a 48x1G+4x10G OF switch and $\$11,200$ for a 48x10G+4x40G device. Based on these considerations, the following formulas



(a) CAPEX per 1G port density (b) CAPEX per 10G port density

Fig. 3: CAPEX for a software, COTS/white-box hardware, and HARMLESS switch as the function of the number of (a) 1G access ports and (b) 10G access ports (prices from 2017).

estimate the CAPEX of hardware switch deployment with a total of x ports:

$$C_{HW}^{1G} = \$2700 \left\lceil \frac{x}{48} \right\rceil, \qquad C_{HW}^{10G} = \$11200 \left\lceil \frac{x}{48} \right\rceil .$$

In case of softswitches, the main CAPEX factor is purchasing servers with a sufficient number of NICs and CPUs. We consider x86-based 1U servers at a bulk price of $\$1,400$ on average, including the motherboard with 1 CPU, 4x1G built-in ports, 3 PCIe (3.0 x8 or x16) slots, memory, disk, power, etc. A server can host up to 3 additional NICs, costing \$160 for 4x1G (Intel i350), \$480 for 4x10G (Intel X710), and \$500 for 2x40G (Intel XL710), which total up to 16 ports per server at $1G$, 12 ports at 10G, and 6 ports at 40G. Note that when a single server cannot provide the required port density another server must be purchased. Furthermore, in most cases the third PCIe slot is hardwired to the second CPU socket, therefore a single CPU server can host up to 12 1G (8 at 10G, 4 at 40G) ports; for more ports per server a second CPU must be installed (hence the last negative term in the below formulas, where the variable $\#extraCPU$ indicates in both the 1G and 10G cases whether the last server is sufficient to serve the rest of the ports w/o an additional CPU). The price of a server CPU ranges between \$200 and $\$7,000$ depending on the CPU class, cache size, clock rate, and power consumption; we calculate with the price of a 6-core Intel E5-2620v3 CPU at \$400.

With this in mind, the following formulas estimate the CAPEX of a software switch deployment with x ports:

$$C_{SW}^{1G} = \$1400 \left\lceil \frac{x}{16} \right\rceil + \$160 \left(\left\lceil \frac{x}{4} \right\rceil - \left\lceil \frac{x}{16} \right\rceil \right) + \$400 \left(\left\lceil \frac{x}{16} \right\rceil - \#extraCPU_{SW}^{1G} \right)$$

$$C_{SW}^{10G} = \$1400 \left\lceil \frac{x}{12} \right\rceil + \$480 \left\lceil \frac{x}{4} \right\rceil + \$400 \left(\left\lceil \frac{x}{12} \right\rceil - \#extraCPU_{SW}^{10G} \right)$$

$$\#extraCPU_{SW}^{1G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 16) < 13 \\ 0, & \text{otherwise} \end{cases}$$

$$\#extraCPU_{SW}^{10G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 12) < 9 \\ 0, & \text{otherwise} \end{cases}$$

For the sake of simplicity, in the price analysis for softswitches and HARMLESS we do not account for OPEX components (e.g., energy consumption, cooling, cabling, rack space occupancy) and we did not take into account the forwarding and trunk port availability a typical COTS device offers; we return to further CAPEX/OPEX issues later. Fig. 3 shows the CAPEX analysis for the purchase of a single hardware-based and software-based SDN switch. The price figures show that hardware switches are more economical at port density 24 and 48, due to software switches needing many expensive NICs and additional servers to match the same number of ports (even though the price advantages level off at high-end switches with 96 ports).

In summary, current market trends suggest that *hardware SDN switches provide high port density at moderate prices but lag behind software switches in scalability, standards-compliance and programmability*.

B. HARMLESS: High port density at low cost

Next, we evaluate HARMLESS as a cost-efficient middle-ground between the two extremes. HARMLESS unifies the advantages of hardware and software switches, by decoupling the raw forwarding functionality from the OpenFlow glue, resulting in an optimal separation of concerns: high port density by legacy devices, while the softswitch component enables implementing the packet processing intelligence in a clear and extensible way. Observe that the softswitch does not need to match the port density of the legacy switch effectively removing the major cost component, NIC prices.

HARMLESS leverages the existing and deployed computing and networking infrastructure, readily available in many prospective SDN deployments like SOHO networks, small and medium sized enterprises, private data centers, campus networks and private clouds. For these use cases, HARMLESS offers an instantaneous SDN transition path with *zero* initial expenditure, apart from the usual practice of server relocation, cabling, etc. Note, however, that the server running the OpenFlow component and the legacy switches do not even need to be co-located; in fact, the OpenFlow logic can be virtualized at a remote site or even delegated to a public cloud at the price of proper traffic forwarding and slightly increased latency.

Even in cases where spare servers or Ethernet switches are not available, purchasing them anew in a HARMLESS setup is still much less costly than purchasing equivalent SDN network gear from commercial suppliers. For the below CAPEX analysis, we assume that the legacy 48x1G+4x10G (or 48x10G+4x40G at 10G access) Ethernet switches are already on stock (if not, add another couple of hundred USD per switch), so only bare-metal servers for the OpenFlow components and 10G NICs (respectively, 40G) need to be purchased. We aggregate 12 access ports to each trunk port, over-committing the uplink at a similar rate as plain Ethernet networks [35], multiplexing up to 144 access ports (72 at 10G) to a single OpenFlow component. Accordingly, the CAPEX for a HARMLESS (HL for short) deployment with x ports are as follows:

$$C_{HL}^{1G} = \$1400 \left\lceil \frac{x}{144} \right\rceil + \$480 \left\lceil \frac{x}{48} \right\rceil + \$400 \left(\left\lceil \frac{x}{144} \right\rceil - \#extraCPU_{HL}^{1G} \right)$$

$$C_{HL}^{10G} = \$1400 \left\lceil \frac{x}{72} \right\rceil + \$500 \left\lceil \frac{x}{24} \right\rceil + \$400 \left(\left\lceil \frac{x}{72} \right\rceil - \#extraCPU_{HL}^{10G} \right)$$

$$\#extraCPU_{HL}^{1G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 144) < 97 \\ 0, & \text{otherwise} \end{cases}$$

$$\#extraCPU_{HL}^{10G} = \begin{cases} 1, & \text{if } 0 < (x \bmod 72) < 49 \\ 0, & \text{otherwise} \end{cases}$$

Similar to software switches, the first term accounts for the server, the second term for the NICs, and the third for the additional CPUs. Using this estimate, the CAPEX analysis in Fig. 3 shows that HARMLESS is by a large margin the most cost-efficient SDN migration option, thanks to the high level of aggregation that reduces the number of costly servers and NICs as compared to pure softswitches, providing one order of magnitude more economical option.

IV. Evaluation

Next we turn to the practical aspects of HARMLESS and compare it against market alternatives. We evaluate the *scalability, standards compliance, data plane performance, latency* in diverse use cases taken from practical networking applications and under different workloads, and we present the results side by side with the *SDN migration cost analysis (CAPEX/OPEX)* for each possible SDN switch option (softswitch, COTS and white-box switches, and HARMLESS). First, we describe our testbed and the measurement methodology, then we present the specific use cases, and finally we present the evaluation results.

A. Testbed and methodology

Our testbed includes two IBM x3550 M5 servers with Intel Xeon E5-2620v3 processors and 16GB of memory running Debian Linux Jessie 8.0/kernel 4.9, each server equipped with an Intel X710 NIC (10G) and Intel XL710 (40G) NIC, respectively. The setup also contains two legacy switches, a `Cisco 3750X` (24x1G + 4x10G) and an `Arista 7048T` (48x1G + 4x10G), three COTS OpenFlow switches, an `HP 3500` (24x1G), an `Extreme X440` (28x1G + 2x10G), and a `Brocade ICX6610` (28x1G + 4x10G), and two white-box switches (`Quanta T1048` and `Edge-Core AS4610`) all supporting OpenFlow v1.3. The COTS switches represent the state-of-the-art in COTS SDN switching as of 2015, while the white-box switches are from the low-end market of 2016. The switches have the following flow table size limitations:

- `HP 3500`: 1 flow table in TCAM with max. 1500 rules, and 4 further logical tables (processed in software);
- `Extreme X440`: 1 flow table in TCAM with max. 255 flow rules, assuming each one has limited length in terms of match fields, and another table for MAC and VLAN matching rules (also in TCAM);
- `Brocade ICX6610`: 1 flow table with max. 3000 rules in TCAM (half if rules match on both L2 and L3 headers), and no further tables.
- `Quanta T1048`: 1 flow table in TCAM with roughly 2000 flow rules and 6 logical tables for tens of thousands of flow rules and different layer matching (e.g., matching on both source and destination MAC address can only be implemented in a logical table).
- `Edge-Core AS4610`: supports multiple flow tables, one of them containing actions, carrying maximum only 3840 flow rules in TCAM (plus $24,576$ and $32,768$ flow rules for exact matching on destination MAC address and destination IP address, respectively)

In each experiment, one of the servers was configured to run NFPA [36], an Intel DPDK *pktgen*-based benchmarking tool, back-to-back with the system-under-test (SUT). For the software switch evaluations the SUT was provisioned on the other IBM server, running a stable version of OVS (v2.7.0) and ESwitch [15], both compiled with a stable Intel DPDK v16.11.1. The hardware switch option was evaluated on each of our COTS switches, while for HARMLESS the OpenFlow component was again configured on the IBM server running OVS (HARMLESS-OVS) or ESwitch (HARMLESS-ESwitch), connected to one of the legacy switches.

The measurements were conducted over synthetic traffic traces, specially tailored to each use case (see below) to contain a configurable number of flows. Note that packets were never dropped intentionally, instead the OpenFlow pipelines contain default catch-all rules to forward unmatched/dropped packets to the external port; our aim was to measure raw throughput and not whether the switches can filter traffic adequately (they can). With this configuration, packet loss only occurs when the SUT becomes a physical bottleneck and therefore the packet rate received at the packet generator is representative of the raw performance. Packets were minimum-sized (i.e., 64 bytes) and Receive Side Scaling (RSS) was turned on in multi-core setups [37]. All measurements were conducted at 40G for at least 60 seconds [38]. At first the *packet rates were measured in a single-core setup*; note that the attainable throughput using a single core and PCIe x8 v3 bus speed is 15 Gbps (22 Mpps) with 64-byte packets; multi-core scalability is studied in a separate measurement round.

1) Use cases: We considered 4 realistic use cases [36], from private data centers to telco gateways. All scenarios will be cast in a single hypothetical service provider's legacy network (see Fig. 4). The setup contains a smaller data center (DC) with 4 racks connected into a CLOS topology with separate L2 domains at the leaves and an L3 domain as the spine [39], an industrial-scale load-balancer [40], and a telco access gateway [41] that aggregates subscribers located behind Customer Endpoints (CEs) [42].

The lower layer of the DC topology represents the *L2* use case, with each top-of-rack (ToR) switch provisioned as a separate *L2* domain; a sample L2 traffic flow is marked with

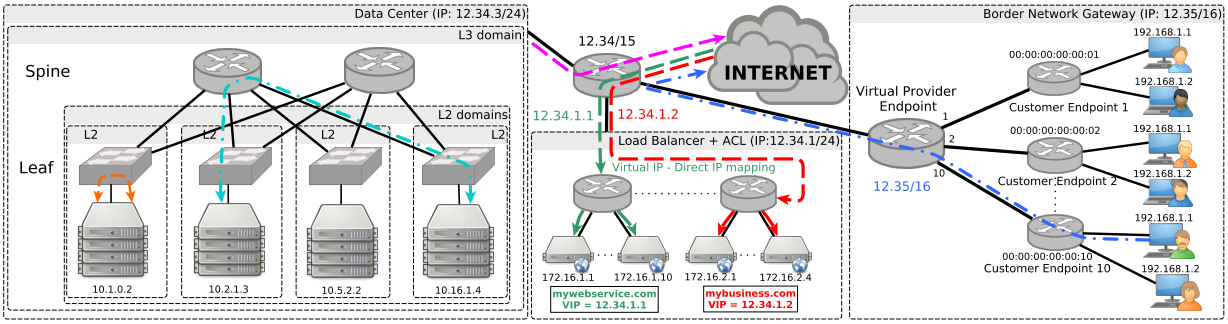


Fig. 4: Use cases in a service provider network.

orange in Fig. 4. While certain data centers may differ in the configuration of L2/L3 domains [43] this use case describes large L2 networks illustratively, to be migrated from traditional 802.1 to SDN with the aim of eliminating dependency on spanning trees and benefiting from centralized control [44].

The *L3* use case embodies the upper layer of the CLOS network interconnecting the L2 islands, a common setup in DCs [39]; a sample traffic is marked with cyan in Fig. 4.

The *load balancer and access control list* use case in the middle of Fig. 4 captures the functionality of a web frontend, balancing incoming web traffic (TCP port 80) for 100 different web services, each available at a unique IP address [36]. During the measurements, traffic traces were crafted so that 70% of packets go to a randomly chosen web service while the rest is filtered at the ACL.

The telco *access gateway* use case [36] on the right hand side of Fig. 4 is the most complex one consisting of a Virtual Provider Endpoint (VPE) that serves Internet access to subscribers located behind Customer Endpoints (CEs). For brevity, we identify CEs with the MAC address and we assume that the operator sets 10 CEs, each serving 20 users provisioned with unique private IP addresses.

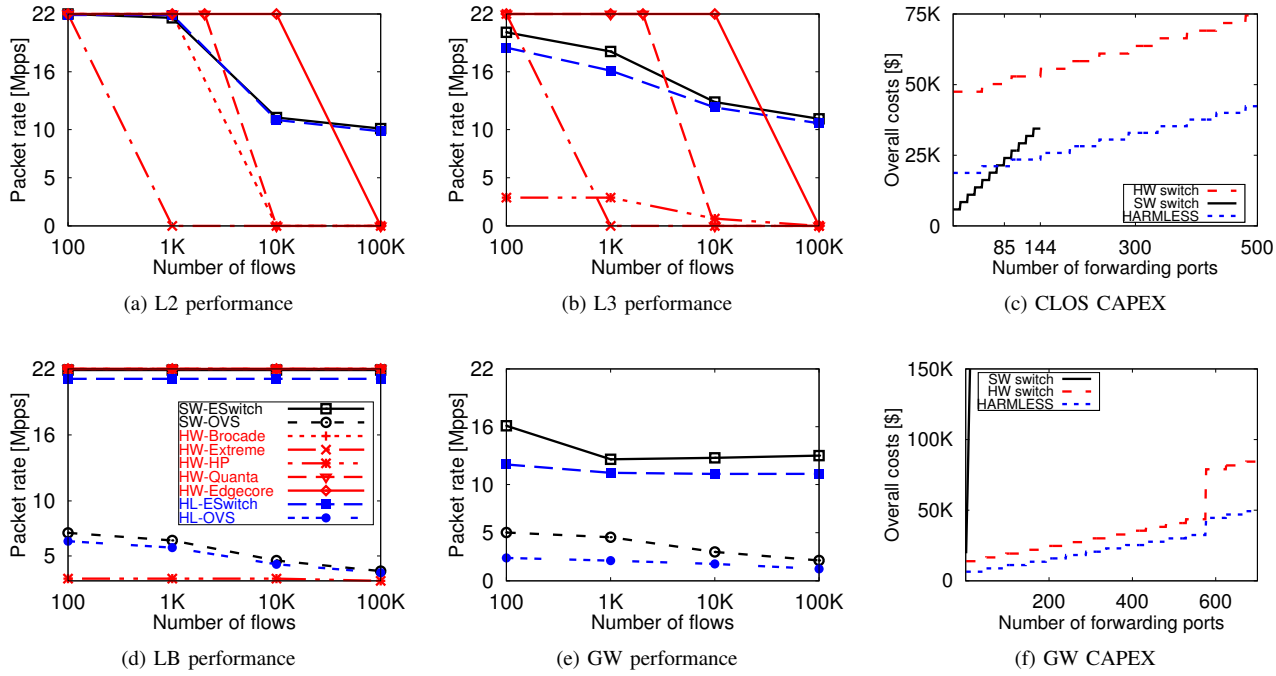
B. Measurement results

1) Scalability and standards-compliance: Configuring the use cases on COTS and white-box switches proved far from trivial, due to the prohibitive flow table sizes and subtle restrictions on flow matching rules. The hardware switches support only a single flow table in TCAM and may or may not provide additional tables in software. Thus, multi-table OpenFlow pipelines had to be tediously collapsed into a single table by hand; in case of the white-box devices their software, e.g., Pica8 PicOS on `Quanta` and ONL+Indigo on `EdgeCore`, do this automatically. Unfortunately, even then the switches rapidly run out of TCAM space because of the flow-state explosion effects for which table collapsing is notorious [10]. In the *access gateway* use case for instance a separate flow entry must be created for every (user, CE, IP prefix) tuple, yielding so many entries that none of the hardware offerings could implement this use case. Furthermore, one has to take into account the ramifications of the chip in each individual switch, e.g., the `HP` switch does not support static matching on MAC addresses, the `Brocade` switch does not support MAC rewrite, the `EdgeCore` switch cannot modify IP fields, only on slow-path. *Current hardware switches do not scale beyond small and medium workloads, and even in that case may require hand-tweaking the OpenFlow pipeline, while softswitches and HARMLESS support very large deployments*.

2) Performance: Fig. 5a, 5b, 5d, and 5e compare the raw packet rate with the hardware switches, the software switches, and HARMLESS measured in the *L2*, *L3*, *load balancer* and *access gateway* use case, respectively. Recall that due to the attainable packet rate of a single CPU core the y axes are scaled up to 22 million packets per seconds (Mpps). Note that for each use case results are reported only for the hardware switches that could handle the use case.

Our observations are as follows. First, as long as hardware OpenFlow switches manage to forward packets purely in the fast path they perform at wirespeed. However, as soon as a hardware switch runs out of TCAM space and forwarding falls back to the software slow path performance plummets. Note, however, that among the devices providing logical tables only `HP` can use TCAM and logical tables at the same time for the same scenario; `Quanta` installs as many flow rules in its TCAM as it can ($2K$), and silently ignores the rest without notifying the controller. For instance, in the *L2* use case the `Extreme` switch could handle 100 flows at line rate ($1K$ and $2K$ flows with the `Brocade` and the `Quanta`, respectively) but could not tackle $1K$ flows at all ($10K$ for the `Brocade` and the `Quanta`). On the other hand, all hardware switches can support the relatively small flow table of the *load balancer* use case adequately (even though the `HP` proved to be slower).

Meanwhile the ESwitch-based OpenFlow softswitch performs close to line rate at small and medium sized workloads and only becomes worse at very large flow tables. Depending on the workload, *the HARMLESS-ESwitch combination attains a performance very close to that of the TCAM-based fast path of hardware switches and the best softswitches*, in the majority of the cases reaches up to 90–95% *and it robustly outperforms the hardware's slow-paths and the* `HP` *switch*. Results with OVS (only presented for the *load balancer* and the *access gateway* use cases for brevity) are much worse, but then again the HARMLESS-OVS mix is very close to pure OVS. This suggests that *the performance of HARMLESS is eminently conditioned on the OpenFlow softswitch component*; here, the HARMLESS-ESwitch combination seems very appealing.



(a) L2 performance (b) L3 performance (c) CLOS CAPEX

(d) LB performance (e) GW performance (f) GW CAPEX

Fig. 5: Raw packet throughput as the function of the workload size in (a) the *L2*, (b) the *L3*, (d) the *load balancer* (LB) and (e) the *access gateway* (GW) use cases, and the CAPEX at different deployment scales for (c) a full CLOS topology that integrates the *L2* and *L3* use cases, and (f) the *access gateway*. Note the common legend for panel (a), (b), (d), and (e) in plot (d): SW: software switches, HW: hardware switches, HL: HARMLESS.

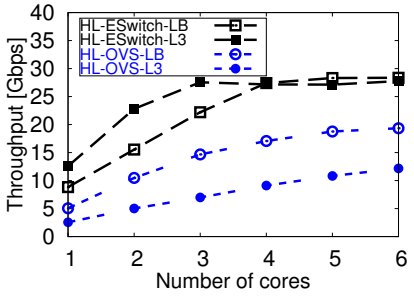


Fig. 6: HARMLESS Multi-core scalability: throughput (Gbps) with ESwitch and OVS (128-byte).

We measured the throughput on multiple CPU cores (Fig. 6) under the larger workloads (namely, in the L3/100K and LB/10K use cases [36]); this time, we use 128-byte packets as the Intel XL710 NIC cannot be saturated with smaller packets [45]. The results indicate that HARMLESS scales to multiple cores linearly, however the HARMLESS-ESwitch mix already achieves its maximum performance (much higher than OVS can attain with 6 cores) with only 4 cores.

3) CAPEX: Fig. 5c compares the CAPEX for a greenfield deployment in the CLOS-based telco DC (the *L2* and *L3* use cases combined) as the function of access port density supported at the ToR switches. Note that due to the different form factors the spine layer scales differently: purchasing four 48x10G hardware switches for the spine incurs a huge initial investment but can then scale to 48 leaf switches economically; in case of software switching, one leaf switch, offering similar aggregation ratios as a typical hardware device provides (e.g., 48x1G vs. 4x10G), mounts 12x1G + 1x10G, thus we only need *one server* with 12x10G capacity as the spine resulting in a small CLOS topology (providing 144 access ports at the most). Observe in Fig. 5c that in contrast to COTS devices and HARMLESS, where we are given 4 spine switches (the most expensive parts of the CLOS topology), the necessity of only one spine server is the only reason why software switches involve less initial investment.

Since in case of HARMLESS the trunk ports of the legacy devices are used to provide the OpenFlow capability, special attention is needed in order to preserve the non-blocking 1:1 over-subscription ratio of the CLOS topology. Therefore, in HARMLESS a spine switch is comprised of a 48x10G+4x40G legacy switch plus a server with two 4x10G NICs for the softswitch component, and 2x100G NIC with an additional CPU for compensating the inherent "loss" of uplink ports resulting in a sum of $4,100 per spine switch (for the NIC we considered the average price of a Mellanox ConnectX-5 NIC of $1,300). Nonetheless, a legacy leaf switch of 48x1G + 4x10G only requires a HARMLESS server with two 4x10G NICs resulting in an average price of $2,350 per leaf switch.

Fig. 5f gives the CAPEX for a telco *access gateway* greenfield SDN deployment in a simple tree topology (Fig. 4) with a depth of 3 consisting of 48x1G forwarding ports at the leafs, 48x10G aggregation switches in the middle, and one switching gear with 40G forwarding ports as the core (we

TABLE I: Latency over in a bridge and in the L2/1K use cases, energy consumption, and rack space.

SDN switch	Latency [μs]		Power [W]		Rack
	Bridge	L2	Min	Max	
SW-ESwitch	238	233	70	230	3U
HW-HP	138	NA	142	616	1U
HW-HP-SW	697	730			
SL-ESwitch	268	265	164	350	1.3U

considered the average price of $21,500 USD) offering an overall 1:1 over-subscription ratio. One can observe that when relying merely on software switches, expenses can easily reach high even for fewer number of ports. On the other hand, in case of the hardware devices and HARMLESS the steep cost steps arrive at 576 forwarding ports: those indicate the price of the 32x40G OpenFlow-enabled core switch, and the three 2x100G NICs for HARMLESS, respectively. Crucially, *in all cases HARMLESS is the most cost-efficient option, supporting roughly the same performance at the fraction of the price*: on average HARMLESS is 2–4 times less expensive than a softswitch-, a COTS-, or a white-box-based deployment, but the price difference can even reach to an order of magnitude. Here, we assume that the legacy Ethernet switches for HARMLESS are on stock; if not, HARMLESS is still 1.5–3 times more cost-efficient due to the economical price tag of commodity Ethernet switches; however, if Ethernet switches and spare servers are available in adequate number then, recall, SDN migration with HARMLESS incurs zero cost.

4) Latency: In order to check whether the additional softswitch in the loop increases the latency of HARMLESS prohibitively (extra latency occurs for inter-port communication only, but not for out-of-switch traffic), we conducted a series of latency measurements in various setups; Table I gives the results for a single port-forwarding rule in the OpenFlow pipeline and for the L2/1K workload. The `HP` switch, when using the TCAM-based fast path, yields roughly 130 μsec delay, but it is much less efficient when it falls back to the software datapath (around 700 μsec). ESwitch's delay is around 230 μsec reliably, with HARMLESS only 10% more thanks to the fast underlying plain Ethernet switch (adding roughly 30–50 μsec to the softswitch latency), but still much faster than the software fallback of the COTS switch. The results indicate that *the additional softswitch does not introduce prohibitive latency in HARMLESS*, just the contrary, its latency is on par, and in some cases even better than, alternatives; accordingly, HARMLESS latency seems sufficient for anything but the most delay-critical applications.

5) OPEX: Below, we extend our analysis with operational costs, which can constitute a significant factor in the total spending. Table I shows an evaluation of two important OPEX components. The energy consumption is estimated from the datasheets of the switches and the CPUs (note that the legacy Ethernet switch used in HARMLESS consumes less power than a full-scale SDN switch). The rack space occupancy is normalized for the standard 48x1G form factor: 1U for a hardware switch, 3U for the three 16x1G servers needed for a 48-port software switch, and for HARMLESS $1U$ for the legacy 48x1G switch and $1U$ for the 12x10G server, but the latter can handle 2 additional legacy switches as well which gives 1.3U normalized to 48 ports overall. Cabling might be more difficult though, since some high-speed uplinks that could otherwise be used for aggregation are allocated for HARMLESS; yet, the flexibility of access port assignment in HARMLESS may be exploited to optimize cabling costs. Overall, *the costs for operating HARMLESS are at the same level as that of the alternatives*.

V. Related Work

SDN migration. The new levels of abstraction, programmability, and logically centralized control are important motivators for deploying SDN [1] in enterprise networks [46], DC fabrics [47], transport networks [48], [49], WANs [3], [5], and Internet exchanges [50]. However, most deployments involve the complete and irreversible overhaul of the existing legacy networking infrastructure. Incremental deployment strategies [8] seek to find a smoother migration path than a flag-day greenfield upgrade [1], [2]. Managing a heterogeneous network, however, can become rather unwieldy due to the potential interference between coexisting legacy and SDN control planes. For example, forwarding loops may be formed due to the legacy control plane masking certain forwarding decisions from the SDN controller [4]. HARMLESS fits into any of these SDN migration paths smoothly, thanks to its dataplane transparency, fine-grained upgrades, and vendor neutrality.

Hybrid SDN. Similar to HARMLESS, the hybrid SDN scheme Panopticon [2] connects legacy device ports to SDN-capable switches using VLAN tagging. The aim in Panopticon is different though: guaranteeing that each forwarding path traverses at least one SDN switch that can exert control over the traffic along that path. On the contrary, HARMLESS is dataplane-transparent and can accommodate any SDN policy without special tweaking. Furthermore, Panopticon needs a nontrivial number of newly purchased SDN switches, while HARMLESS can introduce the *existing* legacy network infrastructure to under SDN control and hence is more economical.

Fibbing [3], [49] endues a legacy network employing a distributed routing protocol with SDN control. However, it is bound by the limitations of destination-based routing, while HARMLESS opens up the full power of SDN.

VI. Conclusion

Recently, SDN has grown out of the "niche status" and found important use in communication networks. However, there still exist areas it has not penetrated, mainly service provider networks and smaller businesses with less technically savvy IT staff. In this paper, we presented HARMLESS, a new SDN switch design to offer an attractive deployment path.

The main idea in HARMLESS is opening up traditional black-box network gear and virtualizing the switch OS in a separate softswitch component. HARMLESS allows an operator to start experimenting with SDN instantaneously: by connecting the trunk port of a legacy Ethernet switch to a

spare x86 server and firing up HARMLESS, an operator can immediately engage with OpenFlow controller programs with zero initial investment. Later, any combination of legacy ports and switches can be connected to the HARMLESS software switch to incrementally reach a full SDN deployment.

HARMLESS realizes an appealing combination of hardware and software switching, with the hard switch providing the port density and the softswitch delivering programmability. Our comprehensive CAPEX analyses on realistic SDN migration scenarios indicate that HARMLESS attains the most economic SDN migration strategy today, with performance close to (90–95%), and in some cases even higher than that of the alternatives. Crucially, HARMLESS is exempt from the dataplane quirks and performance regressions experienced with COTS OpenFlow appliances. With the continuous evolution of software-based switching and general-purpose packet processing solutions, throughput achieved with HARMLESS will likely further improve in the future.

ACKNOWLEDGEMENTS

The work has been supported in part by the European Cooperation in Science and Technology (COST) Action CA 15127: RECODIS - Resilient communication and services; and by the UK Engineering and Physical Sciences Research Council (EPSRC) projects EP/L026015/1, EP/N033957/1, and EP/P004024/1. Project no. PD 121201 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the PD_16 funding scheme.

REFERENCES

[1] Open Networking Foundation, "SDN Migration Considerations and Use Cases," ONF Solution Brief, Nov 2014.
[2] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *USENIX ATC*, 2014, pp. 333–345.
[3] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *SIGCOMM*, 2015, pp. 43–56.
[4] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *SIGCOMM CCR*, vol. 44, no. 2, pp. 70–75, 2014.
[5] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *SIGCOMM*, 2013.
[6] Brocade, "Migrating to SDN: planning for a smooth transition," [Online: https://goo.gl/cgPFTs], 2014.
[7] M. McNickle, "With hybrid SDN deployment, no need for network forklift," TechTarget SearchSDN, 2013.
[8] M. K. Mukerjee, D. Han, S. Seshan, and P. Steenkiste, "Understanding tradeoffs in incremental deployment of new network architectures," in *CoNEXT*, 2013, pp. 271–282.
[9] I. Pepelnjak, "Q&A: Vendor Openflow Limitations," http://blog.ipspace.net/2016/12/q-vendor-openflow-limitations.html, Dec 2016.
[10] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *NSDI*, 2015.
[11] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *PAM*, 2015, pp. 347–359.
[12] I. Pepelnjak, "Table Sizes in OpenFlow Switches," http://blog.ipspace.net/2016/03/table-sizes-in-openflow-switches.html, March 2016.
[13] M. Kuzniar *et al.*, "A SOFT way for OpenFlow switch interoperability testing," in *CoNEXT*, 2012.
[14] A. Pavlidis *et al.*, "Overview of SDN Pilots Description and Findings: Part A," Deliverable D7.1, 2017.
[15] L. Molnár *et al.*, "Dataplane specialization for high-performance openflow software switching," in *Proceedings of ACM SIGCOMM*, 2016, pp. 539–552.
[16] Intel, "Guide: Data plane development kit for linux," Guide, April 2015.
[17] N. Egi *et al.*, "Towards high performance virtual routers on commodity hardware," in *CoNEXT*, 2008, pp. 1–12.
[18] M. Dobrescu *et al.*, "RouteBricks: Exploiting parallelism to scale software routers," in *SOSP*, 2009, pp. 15–28.
[19] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "PISCES: A programmable, protocol-independent software switch," in *SIGCOMM*, 2016, pp. 525–538.
[20] M. Casado *et al.*, "Rethinking packet forwarding hardware," in *HotNets*, 2008.
[21] D. Moon *et al.*, "Bridging the software/hardware forwarding divide," uC Berkeley.
[22] K. Argyraki *et al.*, "Can software routers scale?" in *PRESTO*, 2008, pp. 21–26.
[23] G. Pongrácz, L. Molnár, Z. L. Kis, and Z. Turányi, "Cheap silicon: A myth or reality? picking the right data plane hardware for software defined networking," in *HotSDN*, 2013, pp. 103–108.
[24] J. Gross, A. Lambeth, B. Pfaff, and M. Casado, "The rise of soft switching, Part I, II, III," Network Heresy, 2011.
[25] G. Ferro, "Soft switching fails at scale," EtherealMind, 2011.
[26] N. Gaur, "Fundamentals of vlans: Router on a stick," CCENT/CCNA R&S Study Group, 2014.
[27] Cisco, "Network address translation on a stick," Technical study, Document ID: 6505, 2008.
[28] F. F. Andrew Lunn, Vivien Didelot, "Distributed switch architecture," Netdev Conf 2.1, 2017.
[29] IEEE, "Std 802.1ad - 2005 IEEE Standard for Local and metropolitan area networks – virtual Bridged Local Area Networks, Amendment 4: Provider Bridges"," 2005.
[30] M. Szalay *et al.*, "Harmless: Cost-effective transitioning to sdn," in *Proceedings of the SIGCOMM Posters and Demos*, 2017, pp. 91–93.
[31] Cisco, "Campus network for high availability design guide," Design Guide, 2008.
[32] Juniper, "Campus networks reference architecture," 2010.
[33] N. Foster *et al.*, "Frenetic: a network programming language," in *ICFP*, 2011, pp. 279–291.
[34] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with Pyretic," *USENIX Mag.*, vol. 38, no. 5, 2013.
[35] H. Hudson, "Extending access to the digital economy to rural and developing regions," The MIT Press, 2002.
[36] L. Csikor, M. Szalay, B. Sonkoly, and L. Toka, "NFPA: Network function performance analyzer," in *Proc. of NFV-SDN*, 2015.
[37] Microsoft, "MSDN: Introduction to Receive-Side Scaling," [Online: http://goo.gl/BpoErm, accessed 05-07-2016].
[38] S. Bradner *et al.*, "Benchmarking methodology for network interconnect devices," RFC 2544, 1999.
[39] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *NSDI*, 2014.
[40] R. Gandhi *et al.*, "Duet: Cloud scale load balancing with hardware and software," in *SIGCOMM*, 2014, pp. 27–38.
[41] Intel, "Network function virtualization: Virtualized BRAS with Linux and Intel architecture," https://goo.gl/TVj8co.
[42] S. K. N. Rao, "SDN and its USE-CASES-NV and NFV," White Paper, NEC technologies India Limited, 2014.
[43] Cisco, "Cisco Data Center Infrastructure 2.5 Design Guide," https://goo.gl/kW78VM, Nov 2011.
[44] C. Kim *et al.*, "Floodless in Seattle: a scalable Ethernet architecture for large enterprises," in *SIGCOMM*, 2008, pp. 3–14.
[45] Intel Corporation, "Intel Ethernet Converged Network Adapters XL710 10/40 GbE," Datasheet, 2015.
[46] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *HotSDN*, 2012, pp. 115–120.
[47] N. Mysore *et al.*, "PortLand: a scalable fault-tolerant Layer 2 data center network fabric," *SIGCOMM CCR*, vol. 39, no. 4, pp. 39–50, 2009.
[48] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *CoNEXT*, 2013, pp. 13–24.
[49] M. Chiesa, G. Rétvári, and M. Schapira, "Lying your way to better traffic engineering," in *CoNEXT*, 2016.
[50] A. Gupta *et al.*, "SDX: a software defined Internet Exchange," in *SIGCOMM*, 2014, pp. 551–562.

I DPID It My Way!
A Covert Timing Channel in Software-Defined Networks

Robert Krösche* Kashyap Thimmaraju* Liron Schiff† Stefan Schmid‡§*
*TU Berlin †GuardiCore Labs ‡University of Vienna §Aalborg University

***Abstract*—Software-defined networking is considered a promising new paradigm, enabling more reliable and formally verifiable communication networks. However, this paper shows that the separation of the control plane from the data plane, which lies at the heart of Software-Defined Networks (SDNs), can be exploited for covert channels based on SDN Teleportation, even when the data planes are physically disconnected.**

This paper describes the theoretical model and design of our covert timing channel based on SDN Teleportation. We implement our covert channel using a popular SDN switch, Open vSwitch, and a popular SDN controller, ONOS. Our evaluation of the prototype shows that even under load at the controller, throughput rates of 20 bits per second are possible, with a communication accuracy of approximately 90%. We also discuss techniques to increase the throughput further.

1. Introduction

In the recent years computer networks have undergone a transformation to overcome *ossification* [1]. Existing communication protocols and architectures were unable to meet the increasingly stringent requirements, e.g., in terms of performance but also dependability, of growing networks such as data center networks and wide area networks [2].

One of the answers to the *ossification* problem is what is now known as Software-Defined Networks (SDN) which is the separation (and consolidation) of the network control plane from the data plane. SDNs promises innovation, reduced cost and better manageability [3].

As of today, we witness an increasing interest in SDN not only in academia and the industry but also by governments [4]. Several open-source SDN projects have gained wide-spread adoption by the community, e.g., Open vSwitch and OpenDayLight are a part of the Linux foundation. Hardware vendors are also adopting the SDN paradigm and shipping software programmable network cards [5].

While the literature has demonstrated well how an SDN can overcome the shortcomings of traditional networks and while SDNs are rapidly gaining traction, researchers have also identified new security challenges they introduce. For example, Hong et al. [6], and Dhawan et al. [7] identified ways for an attacker to spoof the controller's view of the network topology. Jero et al. [8] identified a weakness in the way controllers bind *network identifiers* allowing an attacker to conduct a man-in-the-middle attack.

Those papers show that attacks on the controller can easily occur from the data plane. The assumption that the data plane can be compromised, e.g., via trojans, or software exploits, is not far fetched. For example, Thimmaraju et al. [9] demonstrated the simplicity of compromising the data plane of an SDN-based cloud system.

The SDN controller may also be exploited for *teleportation*, e.g., malicious switches or hosts can communicate via the control plane and circumvent data plane security mechanisms [10] to exfiltrate sensitive information. Teleportation can also be exploited by physically disconnected switches, e.g., switches in different geographic locations. More importantly, teleportation is inherent to an SDN. Among the teleportation techniques identified [10], out-of-band forwarding, flow reconfiguration and switch identification, only out-of-band forwarding has been explored in the literature [10]. Switch identification and flow reconfiguration were described as a *Rendezvous Protocol*.

Hence in this paper, we go beyond the initial intention of switch identification teleportation by describing how it can also be used for covert communication: malicious switches can transfer a 2048 byte *RSA private key file* in ∼13 minutes. In particular, we design, develop and evaluate a time-based covert channel using the switch identification teleportation.

Our Contributions: We describe the state machine of switch identification and model it in terms of time delays. We then design a covert timing channel using our model. We prototype our design and evaluate its performance and accuracy. Finally, our study of the OpenFlow handshake leads us to the observation that it is currently insecure. The vulnerability received *CVE-2018-1000155* and mitigations have been announced.

Novelty and Related Work: To the best of our knowledge, this is the first paper that describes a covert timing channel in an SDN, and OpenFlow-based network in particular. We are only aware of one other paper dealing with covert channels in SDN, which is however very different in nature: Hu et al. [11] proposed to use SDN to improve the detection of storage covert channels that use the TCP flags for covert communication. More generally, the study of covert channels dates back to the 80's when Simmons [12] introduced the "Prisoners Problem" and the *subliminal channel*. Network based covert channels in local area networks were introduced by Girling [13], wherein a covert channel based on the inter frame delay was proposed. Handel et al. [14] conducted an extensive study on viable covert channels within the OSI

ISBN 978-3-903176-08-9

networking model. A covert channel based on sending an IP packet or not in a time interval was demonstrated by Cabuk et al. [15]. More recently, Tahir et al. [16], designed and developed Sneak-Peek, a high speed covert channel in data center networks. Their covert channel also utilizes a delay mechanism wherein the sender's *flow* introduces a delay into the receivers *flow* over the same network link thereby covertly communicating information based on the delay measured by the receiver.

Paper Organization: Section 2 introduces our threat model followed by a description of our covert channel in Section 3. We describe the key challenges in Section 4 followed by our evaluation in Section 5. After a brief discussion is Section 6, we conclude in Section 7.

2. Threat Model

We consider a threat model where OpenFlow switches can be malicious. For example, the attacker compromises the switch by exploiting a (parsing) vulnerability [9], or the attacker compromises the supply-chain and introduces hardware trojans into the switches [17]. The objective of the malicious switches is to *covertly communicate* information, e.g., private keys, confidential meta-data, attack coordination, even in the presence of security mechanisms, e.g., firewalls, in the data and control plane. The attacker chooses *covert* communication instead of *overt* to persist and remain undetected in the network, e.g., an Advanced Persistent Threat (APT).

We place no restrictions on what a malicious switch can and cannot do. For instance, the switch can send fake OpenFlow messages, it can arbitrarily deviate from the OpenFlow specification, and it can even use multiple identifiers, all at the risk of being detected. However, the position of the malicious switches in the network is not under the control of the attacker. For example, the malicious switches are separated by a firewall that prevents bi-directional communication, or the switches are physically disconnected (geographically separated). However, the malicious switches are connected to the same logically centralized controller. In order to covertly communicate, the malicious switches have been programmed to recognize some data and timing patterns.

The OpenFlow controller and its applications on the other hand are trusted entities and are available to the switches, e.g., they are based on static and dynamic program analyses. The OpenFlow channel is reliable and may be encrypted.

3. A Covert Channel using Teleportation

Covert channels are communication channels that were not designed with the intention for communication [18]. They can be used to bypass security policies, thereby leading to unauthorized information disclosure [19]. A covert timing channel is one wherein a sender and receiver "use an ordering or temporal relationship among accesses to a shared resource" [18] to covertly communicate with each other. In the following we describe how switch identification

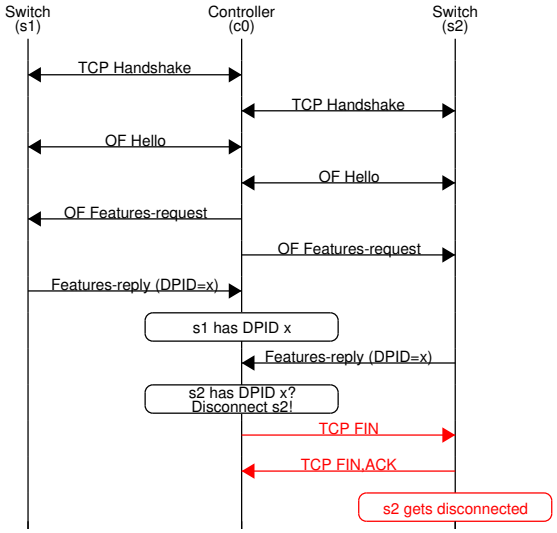


Figure 1: Message sequence pattern for the OpenFlow handshake and *switch identification* teleportation when the controller denies the second switch a connection.

teleportation can be used as a covert timing channel in a software-defined network using the OpenFlow protocol.

Switch Identification Teleportation: In an OpenFlow network, the switch typically initiates a TCP connection with the OpenFlow controller as shown in Fig. 1. If TLS/SSL is configured, the connection is further authenticated and subsequent messages exchanged are encrypted as well. Once the transport connection is established, the switch sends the controller an OpenFlow *Hello* message. The controller responds with a *Hello* message. These messages are used to negotiate the OpenFlow version to be used. Next, the controller sends the switch a *Features-Request* message. The switch replies with a *Features-Reply* message. The *Features-Reply* message includes a *Datapath ID* (DPID) field that uniquely identifies the switch to the controller. After processing the *Features-Reply* message, the OpenFlow connection is considered established, and ready for operation [20].

A fundamental requirement of an SDN is for the controller to uniquely identify the switches in the network which is achieved by the switch providing "identity" information, e.g., DPID in the *Features-Reply* message, to the controller. *Switch identification teleportation* is the outcome of two switches connecting to the same logical controller using the same DPID [10]. We have identified 4 possible outcomes when this occurs in OpenFlow: i) The controller denies a connection with the second switch; ii) The controller accepts the connection with the second switch, and terminates the first switch's connection; iii) The controller accepts connections for both switches; iv) The controller accepts connections for both switches, however, each switch receives a different *Role-request* message. Only in outcomes i, ii and iv can the malicious switches infer if the DPID it used is already in use by another switch. The message sequence pattern for the OpenFlow handshake and outcome i is shown in Fig. 1.

3.1. Single Bit Transfer

From the message sequence pattern in Fig. 1, switch $s2$ can infer a binary value of 1 if it gets disconnected, and a binary value of 0 if it is able to connect, thereby received one bit of data. We can precisely describe the states and transitions to transfer one bit value as state machines for the sender and receiver resp. Additionally, we can precisely describe a time-based model to transfer one bit value that can be leveraged to design a channel to transfer multiple bits. In the following we describe the state transition model and time model to transfer one bit. Following that, we describe our algorithms to transfer multiple bits.

3.1.1. State Transition Model. The state transition model for switch identification involves a *sender* and *receiver*. As the names imply, the sender sends a binary bit value by either connecting to the controller or not. Similarly, the receiver receives a binary bit value by detecting whether its OpenFlow connection to the controller is allowed or denied.

In our model, we make the following assumptions. We assume that the sender and receiver use an a priori agreed upon DPID (one that is not used in the network), a time to connect to the same OpenFlow controller and a time interval Δ. Δ, is the total time the sender and receiver use to send and receive resp. a bit value. The sender and receiver have synchronized their clocks. We discuss synchronization further in Sec. 4.1. The receiver in particular, is always able to connect to the controller a short δ_{offset} time after the sender. The controller, behaves according to outcome i (see Switch Identification Teleportation). The receiver infers a binary bit value of 1, if its OpenFlow connection is denied, i.e., the sender connected to the controller before the receiver. The receiver infers a binary bit value of 0, if its OpenFlow connection is accepted, i.e., the sender did not connect to the controller.

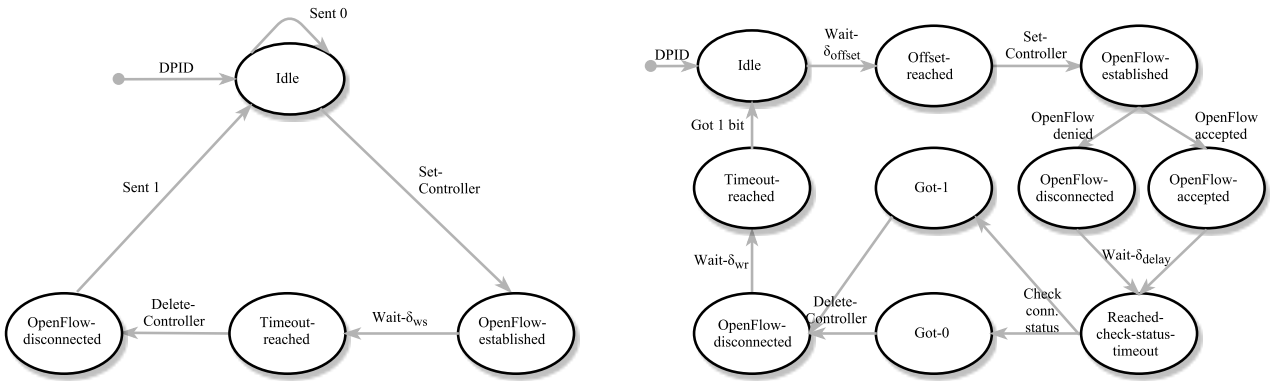
The sending and receiving of bit information can be described in more detail by defining a set of states and transitions for the sender and receiver resp., as shown in Fig. 2. The sender *starts* data transmission with an agreed upon DPID, by entering into the *Idle* state. To send a 0, it simply remains in the *Idle* state. To send a 1, it transitions to the *OpenFlow-established* state via the *Set-Controller* transition. *Set-Controller* involves initializing internal objects, e.g., $rconn$ and $vconn$ data structures in Open vSwitch, in order to initiate a transport (e.g., TCP) connection to the controller at a specific IP and port address. It also involves establishing the TCP and OpenFlow connection with the controller. Once the OpenFlow connection is established, the sender waits for a timeout δ_{ws}, to move into the *Timeout-reached* state. From there, the sender enters into the *OpenFlow-disconnected* state by tearing down the TCP and OpenFlow connection, and deleting its controller information. From thereon, the sender completes a bit transfer by entering back into the *Idle* state. The sender's state diagram is depicted in Fig. 2a.

The receiver also starts with the same DPID to enter into the *Idle* state. Unlike the sender, the receiver must always attempt to connect to the controller to receive a 0 or a 1. It waits for δ_{offset} time to enter the *Offset-reached* state before it sets the controller to enter into the *OpenFlow-established* state, similar to the sender. If the receiver's OpenFlow connection is denied, it will enter into the *OpenFlow-disconnected* state resulting in its OpenFlow and transport connection being terminated. If the receiver's OpenFlow connection is accepted, it will enter into the *OpenFlow-accepted* state resulting in its OpenFlow connection being sustained. Regardless of the outcome, the receiver waits δ_{delay} time, thereby transitioning to the *Reached-check-status-timeout* state. From there, the receiver checks the OpenFlow connection status. It enters the *Got-1* state if it was disconnected, i.e., it got a 1. It enters the *Got-0* state if it was accepted, i.e., it got a 0. From there on the receiver deletes its controller information, resulting in the OpenFlow and transport connection being torn down if it is still present. Depending on the value of Δ, there may still be time left, hence the receiver waits δ_{wr}, till the end of time interval, to enter the *Timeout-reached* state. It then completes the reception by moving back into the *Idle* state. The state diagram for the receiver is shown in Fig. 2b.

3.1.2. Transition Delays. To leverage switch identification as a covert timing channel we must first establish the time it takes for the sender to send a 1—as sending a 0 requires the sender to remain in the *Idle* state—and the receiver to receive a bit value. We define a *time interval* Δ, as the time the sender and receiver use to send and receive resp. a binary bit value.

Δ comprises of the several state transitions described for the sender and receiver (Sec. 3.1.1). We can construct a time-based model by considering the transitions as delays or timeouts for the sender and receiver that can be used to analyze the performance of our covert channel. In the following we define the various delays and timeouts for the sender and receiver state transitions.

1) δ_s: The time the sender takes to send a binary bit value.
2) δ_r: The time the receiver takes to receive a binary bit value.
3) δ_{sc}: The time to transition from the *Idle* state to the *OpenFlow-established* state.
4) δ_{dc}: The time to move from the *OpenFlow-established* state to *OpenFlow-disconnected* state.
5) δ_{offset}: A timeout value the receiver waits before it sets the controller.
6) $\delta_{of\text{-}deny}$: The time to move from *OpenFlow-established* to *OpenFlow-disconnected* when the connection is denied.
7) δ_{delay}: A timeout value the receiver waits before it checks the OpenFlow connection status.
8) $\delta_{chk\text{-}conn}$: The time the receiver takes to determine a 0 or 1 by checking the OpenFlow connection status.
9) $\delta_{ws} = \Delta - \delta_s$: A timeout value the sender waits before moving from the *OpenFlow-established* state to *OpenFlow-disconnected*.



(a) Sender

(b) Receiver

Figure 2: State diagram for the sender and receiver to send/receive one binary value.

10) $\delta_{wr} = \Delta - \delta_r$: A timeout value the receiver waits before moving from the *OpenFlow-disconnected* state to the *Idle* state.

Using the above definitions, we can now compute the time to send and receive a 0 or 1. The total time to send a 0 or 1 is shown in Eq. 1. As we can see, it takes more time to send a 1 compared to a 0. In Eq. 2, we can see the time it takes to receive a 0 or a 1. In particular, the different delay is $\delta_{of\text{-}deny}$ for the 1. For the sender and receiver to operate correctly, we require the inequality shown in Eq. 3 to hold, i.e., the time interval Δ must not be less than the total time to send or receive a binary bit value.

Additionally, for the receiver to correctly detect a 0 and 1, we require the inequalities as shown in Eq. 4 and 5 to hold. The former equation states that δ_{offset} must be greater than the time it takes for the sender to enter the *OF-established* state. This is to ensure that the receiver does not connect before the sender when the sender wants to send a 1. The latter equation states that the minimum amount of time it can wait before checking the OpenFlow connection status is 0, and the maximum time it can wait depends on the time interval, the time elapsed so far, and the time for the remaining transitions to complete. The δ_{delay} gives the receiver the flexibility of waiting for some amount of time before checking the status of the OpenFlow connection. For example, checking the connection status at $\Delta/2$, i.e., at the middle of the time interval, may be better than checking it at $\Delta/4$. Hence, the receiver can set δ_{delay} such that, the OpenFlow connection status is checked at a point where the connection is most stable.

$$\delta_s = \begin{cases} 0, & \text{to send } 0 \\ \delta_{sc} + \delta_{dc}, & \text{to send } 1 \end{cases} \tag{1}$$

$$\delta_r = \begin{cases} \delta_{offset} + \delta_{sc} + \delta_{delay} \\ +\delta_{chk\text{-}conn} + \delta_{dc}, & \text{to get } 0 \\ \delta_{offset} + \delta_{sc} + \delta_{of\text{-}deny} \\ +\delta_{delay} + \delta_{chk\text{-}conn} + \delta_{dc}, & \text{to get } 1 \end{cases} \tag{2}$$

$$\delta_s \leq \delta_r \leq \Delta \tag{3}$$

$$\delta_{offset} \geq \delta_{sc} \tag{4}$$

$$\begin{aligned} 0 \leq \delta_{delay} \leq \Delta - (\delta_{offset} + \delta_{sc} \\ +\delta_{of\text{-}deny} + \delta_{chk\text{-}conn} + \delta_{dc}) \end{aligned} \tag{5}$$

3.2. From One Bit to Multiple Bits

Until now, we have described how the sender can transmit only a single bit value to the receiver. To receive the single bit value, the sender and receiver need to be synchronized, i.e., the sender and receiver must know the exact time at which the time interval Δ begins and ends. To this end, we assume the sender and receiver synchronize their clocks using the same network time protocol (NTP) time server. Furthermore, we assume the sender and receiver a priori agree upon specific times at which they will initiate their covert communication.

In order to be useful, a covert channel should provide a sender with the ability to transmit several kilobytes of data, e.g., an RSA private key file. Accordingly, in the following, we extend our discussion from a single bit transmission to multiple bits. First, the sender and receiver must agree upon an encoding/decoding scheme, e.g., ASCII. Second, they must also agree upon a method to signal the *start* and *end* of a message. To do so, we use a frame-based transmission method. In particular, the sender encodes a message M into into *frames* F, of length Fl, and transmits the frames. The receiver, decodes each frame received to obtain the sent message.

For *simplicity*, we consider a frame with at least one *SoF* (Start of Frame) bit, and at least seven *data* bits (e.g., ASCII characters can be represented in 7 bits). The SoF bit is used by the sender to signal the receiver that a frame transmission begins which is followed by data bits. We assume that the *SoF* bit is a binary 1, and if the receiver gets this value at the

Algorithm 1 To send binary data as frames.

Require: Message M, Frame-length Fl, Frames F, Time-interval Δ, Start-time t

```
1: initialize(sender)
2: for frame ∈ F do
3:     set-controller                          ▷ Send SoF bit
4:     Wait δ_ws
5:     for bit ∈ frame do
6:         if (bit==0) then
7:             delete-controller               ▷ Send 0
8:         else
9:             set-controller                  ▷ Send 1
10:        Wait δ_ws
11:    delete-controller
```

Algorithm 2 To receive binary data as frames.

Require: Frame-length Fl, Time-interval Δ, Start-time t

```
1: initialize(receiver)
2: while End of message not received do
3:     Wait δ_offset
4:     set-controller                          ▷ Receive SoF bit
5:     Wait δ_delay
6:     Check OpenFlow connection state
7:     if OpenFlow denied then                 ▷ Got SoF bit
8:         Wait δ_wr
9:         for bit ∈ Fl do
10:            set-controller                  ▷ Get data bit
11:            Wait δ_delay
12:            Check OpenFlow connection state
13:            if OpenFlow accepted then
14:                frame += "0"                ▷ Got 0
15:            else frame += "1"               ▷ Got 1
16:            delete-controller
17:            Wait δ_wr
18:        if frame =="0000000" then
19:            End of message received
20:            Break                           ▷ Terminate reception
21:        else
22:            M+ = frame                      ▷ Append frame to message
```

agreed upon time and time interval, it will begin receiving data bits. The data bits can be 0 or 1 depending on how the message is encoded. To indicate the end of a message, the sender sends a frame with all the data bits as 0. When the receiver receives such a frame, it will terminate execution. The above steps are specified as algorithms for the sender and receiver in Alg. 1 and 2 resp.

The sender's algorithm, accepts several inputs, e.g., M is the message to be transmitted, Fl is the frame length, e.g., 8, F is the list of frames that are to be sent, Δ is the time interval, and t is the transmission start-time. The input values for the receiver are the same frame length, time interval and start-time as the sender.

For every frame to be sent, the sender first sends a *SoF* bit for that frame by connecting to the controller. Similarly the receiver waits for δ_{offset} time before attempting to receive the *SoF* bit. If its connection is denied, it will begin receiving data bits. After sending the *SoF* bit, the sender sends data bits: if sending a 0, it disconnects from the controller, if sending a 1, it connects to the controller. It then waits till the end of the timing interval before sending the next data bit. The receiver detects the data bits in a frame by connecting to the controller, and waiting for δ_{delay} time before checking whether its OpenFlow connection was allowed or not. If the connection was accepted, it will append a 0 to the data bits received in the frame, otherwise it will append a 1. The receiver then deletes the controller, and then waits δ_{wr}, i.e., till the end of the time interval before connecting to the controller again.

Once the sender has sent the data bits of a frame, it will wait δ_{ws} time, i.e., for the next time interval to send the next frame. The receiver detects the end of a message when it has received a frame with all the data bits zeroed, thereby terminating the while loop at the receiver. The receiver can then decode the binary data to reveal the message sent.

4. Design and Performance Challenges

Our covert channel design requires us to overcome several non-trivial challenges. Hence, we discuss the most important challenges that affects our design in this section before transitioning to our implementation. We also cast light on factors that affect the performance of our design.

4.1. Synchronization

One of the main problems in designing a covert timing channel is synchronization. Lack of synchronization can lead to the receiver obtaining inaccurate information, thereby reducing the accuracy of the channel. The sender and receiver must share a reference clock to ensure that the the algorithms start at the same time. To this end, we use NTP (as it easily available for today's popular operating systems) and the same NTP server to synchronize the clocks of the sender and receiver to achieve at least millisecond accuracy [21]. Since the sender and receiver clocks can slowly drift apart their clocks must be periodically synchronized with the same NTP server.

When the clocks are synchronized, the *SoF* bit(s) in each frame sent synchronizes the receiver with the sender enabling the receiver to obtain the data bits. During the transmission of a frame, we introduce the δ_{ws} and δ_{wr} times for the sender and receiver resp. at the end of a time interval for synchronization across time intervals in a frame. Furthermore, between frames the sender and receiver can synchronize again by waiting, for example for the next second. This *inter frame delay* adds another layer of synchronization to enable the sender and receiver to send and receive resp. the *SoF* bit(s) accurately.

4.2. Determining the Time Interval Δ and Delays

The time interval in which the sender and receiver send and receive a bit leads to the achievable throughput of the channel. As the time interval reduces, the probability of an error occurring increases, e.g., the receiver may check the connection status before receiving the TCP FIN from the controller. Furthermore, system and network artefacts can

non-deterministically influence the state transitions resulting in errors. Hence, the challenge here is to determine a time interval as small as possible within an acceptable level of accuracy ($\geq 95\%$). We empirically identify suitable time intervals in Sec. 5 based on our prototype implementation. However, in the real-world, the channel would have to start with a programmed value, e.g., 1s, and later be negotiated.

Recall Sec. 3.1.2, there are several delays involved in our timing channel. The delays for one network system, may not be applicable elsewhere. Delays such δ_{sc}, δ_{dc}, $\delta_{of\text{-}deny}$, and $\delta_{chk\text{-}conn}$, depend on the system and network conditions. Moreover, they are not under the control of the sender/receiver. The timeouts δ_{offset} and δ_{delay} although bounded (see Eq. 4 and 5 resp.) can be tuned by the receiver. Hence, we evaluate 3 different δ_{delay} values in Sec. 5.

4.3. Frame-based Transmission

Our design uses a frame-based method to transfer data from the sender to the receiver. The smallest frame size we consider is 8 bits long: 1 *SoF* bit and 7 data bits. The size of this frame can change, e.g., we can send 14 or 28 data bits as well. Sending more data bits in a frame reduces the overhead of sending the *SoF* bit. We can also increase the number of *SoF* bits to ensure the receiver can get the data bits. However, increasing the number of bits in a frame increases the probability of errors within a frame. We do not consider error correction in our design although it can be introduced, e.g., using Hamming codes. However, we do include a minimal set of error detections at the receiver which we describe next.

Receiver misses the start bit of the frame: Several reasons can affect the receiver from missing the *SoF* bit of a frame. In such cases the receiver simply remains idle for the remainder of the time that is necessary to transmit an entire frame.

End of Transmission: For simplicity, the sender indicates the end of transmission via a special *EoM* (End of Message) frame. This design choice comes with a couple of challenges for the receiver to correctly terminate. First, if the receiver misses the *SoF* bit of the *EoM* frame, then it will continue to expect to receive frames. To address this problem, we define a threshold number of consecutive frames, e.g., 5, the receiver does not receive beyond which the receiver terminates reception. Second, the receiver can incorrectly detect a 1 as a 0 due to synchronization issues for example. As a result, the receiver may detect the *EoM* prematurely and stop receiving data even though the sender continues to send data. We cannot address this case as it is a limitation of our design to not include the length of the message to be received.

4.4. Influence of the Controller

The OpenFlow controller that is used to covertly communicate is beyond the control of the sender and receiver. Hence, the accuracy and performance of our channel is limited by the controller that operates the OpenFlow network.

Load on the Controller: Typically, there are more switches connected to the controller than just the sender and the receiver of the covert channel. If the communication between the benign switches and the controller is frequent and voluminous, the sender and receiver will experience non-deterministic delays in connecting/disconnecting (δ_{sc}, δ_{dc} and $\delta_{of\text{-}deny}$) to the controller, thereby reducing the performance (throughput and accuracy) of the channel.

Controller Architecture: The system and software architecture of the controller also influences our design. For example, the controller could be single threaded or multi-threaded. The former can lead to long delays, whereas the latter can lead to non-determinism due to the scheduler.

Path to the Controller: Network paths not under the control of the sender and receiver can influence the performance of our channel. For example, buffers in switches can be filled up by other network packets resulting in packet loss and hence errors in the received bits.

5. Evaluation

To obtain deeper insights and validate our expectations of our covert channel, we prototyped our design using Open vSwitch [22] and ONOS [23]. Furthermore, we designed a set of experiments based on the challenges described in the previous section to characterize the performance of our channel. We begin with a brief description of our implementation, and then describe the experiments.

5.1. Implementation

We used Open vSwitch (OvS) as our sender and receiver OpenFlow switches. We only modified the (OpenFlow) connection handling of OvS so that after it disconnects from the controller, it waits for 4 seconds to reconnect. To set/delete controller information, and configure the DPID, we used the *ovs-vsctl* tool that ships with OvS. We then implemented the sender and receiver algorithms (Alg. 1 and 2) as python scripts. In doing so, we traded performance for *simplicity* which we consider acceptable for the sake of prototyping and evaluation. Our implementation is only meant to demonstrate the *feasibility* of our attack.

We synchronized the system clocks of the sender and receiver using our university's NTP time server. To encode and decode the messages, we used the ASCII scheme. We implemented an adaptive *inter-frame delay* synchronization scheme in which the sender sends a frame only at the start of the next second.

5.2. Setup and Methodology

Our evaluation setup comprised of three (sender, receiver and controller) Dell PowerEdge 2950 servers with 4 core Intel(R) Xeon(TM) CPU 3.73GHz processors and 16 GB of RAM each. The sender and receiver were directly connected to the controller. For OpenFlow load generation, we used a fourth server running directly connected to the controller.

All these servers used dedicated ports to connect to a management switch that was used for orchestration from a fifth server to conduct the evaluation. All systems ran Ubuntu 14.04.5 LTS. For the sender and receiver, we used Open vSwitch 2.7. For the controller, we used ONOS 1.10.2.

Based on our covert timing channel design the objectives of the evaluation are the following. First, we want to establish time intervals that achieve high accuracy and throughput. Second, we want to determine the influence the frame length has on the accuracy, e.g., do shorter frames have fewer errors than longer frames? Third, we want to measure the influence of δ_{delay} on the accuracy and throughput of our channel e.g., is there a δ_{delay} value for which the time interval can be smaller? Finally, we want to measure the accuracy of our channel when there is load on the controller.

The general methodology we undertake is the following. The controller runs ONOS with the default applications activated. We program the sender and receiver with a specific start time t, time interval Δ, offset $\delta_{offset} = 5\ ms$, check the connection status at $\Delta/2\ ms$ and frame length Fl. The sender then sends a 64 byte message M_s and the receiver receives a message M_r. We then restart ONOS and OvS, and clean up the OvS database before we repeat the measurement. We collect ten such measurements for the configured values. We measure accuracy as the similarity between M_r and M_s using the *edit distance* or *Levenshtein distance* [24].

For load on the controller, we use OFCProbe [25] as our OpenFlow topology and packet generator. We configure OFCProbe to emulate 20 switches that trigger Packet-Ins to the controller following a Poisson distribution (λ=1). After OFCProbe has started the Packet-in generation, we wait for one minute before we start the sender and receiver, to avoid any warm-up effects from OFCProbe and ONOS.

5.3. Experiments

Following the aforementioned methodology, we now describe the experiments and their results.

Effect of Timing Interval Δ: We set the frame length $Fl = 7$, and measure the accuracy for time intervals from 30 ms up to 100 ms. The results are shown in Fig. 3.

The results depict that our channel can achieve nearly 100% accuracy for time intervals greater than 60 ms when there is no load on the controller. For $\Delta = 60\ ms$, we have a throughput of approximately 16.67 bps. What we can also see is that as the time interval increases the accuracy increases, which is what we expected. Another distinct observation is that for the values configured, our channel cannot operate below 40 ms because the receiver gets the *EoM* prematurely, (it detects only 0 in the data bits).

Effect of Frame Length Fl: To measure the influence of the frame length on the accuracy we chose the following values: 7, 14 and 28. Note that these values represent the number of data bits in the frame, i.e., 1, 2, and 4 ASCII characters resp. We use only one *SoF* bit in the frame. We repeat the measurements for time intervals from 30-100 ms. The results from this experiment are depicted in Fig. 3.

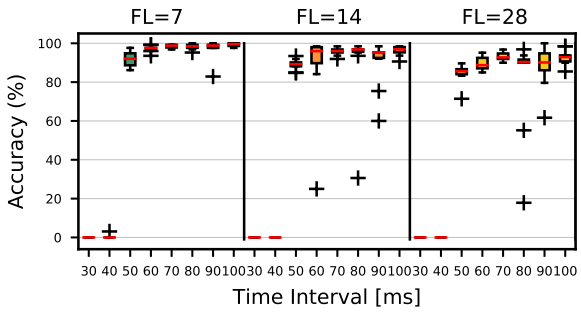


Figure 3: Channel accuracy for time intervals 30-100 ms, and frame lengths 7, 14 and 28 when $\delta_{offset} = 5ms$, OpenFlow status is checked at $\Delta/2$, and there is no load on the controller.

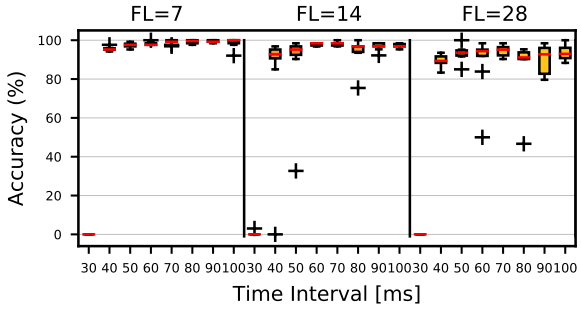


Figure 4: Channel accuracy for time intervals 30-100 ms, and frame lengths 7, 14 and 28 when $\delta_{offset} = 5ms$, OpenFlow status is checked at $2\Delta/3$, and there is no load on the controller.

Indeed, the frame lengths we used show us that as the frame length increases the accuracy drops. Longer frame lengths result in fewer frames but more data per frame being sent. Hence, if the receiver misses the *SoF* bit for $Fl = 14$, it misses twice as many characters compared to $Fl = 7$. Moreover, the chance of incorrect bit detection (bit-flips) increases with larger frames. We analyzed the errors and observed that indeed as the frame length increases, the number of bit-flips increase, and the number of missed *SoF* bits also increase. To address the problem of missing the start bit we can introduce redundant *SoF* bits.

Effect of δ_{delay} in Checking Connection Status: We now investigate how δ_{delay} influences the throughput and accuracy of our channel. Recall from Sec. 3.1.2 that this value is the time the receiver waits before it checks the status of the OpenFlow connection. Until now, we checked the connection status at $\Delta/2$. Hence, in this experiment we check the connection status at $2\Delta/3$ and $\Delta/3$ for frame lengths 7, 14 and 28, and time intervals 30-100 ms. The results for $2\Delta/3$ and $\Delta/3$ are shown in Fig. 4 and 5 resp.

When we check the status at $2\Delta/3$, the 40 ms time interval operates at nearly 100 % accuracy. Moreover, the accuracy for this δ_{delay} value performs better compared to

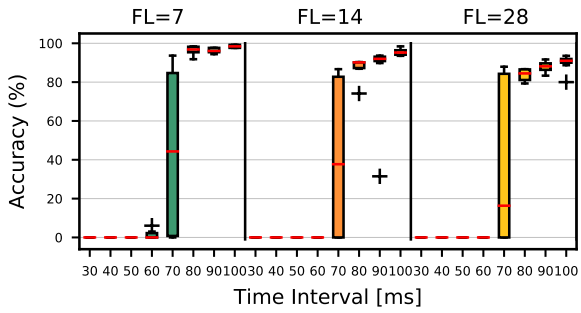


Figure 5: Channel accuracy for time intervals 30-100 ms, and frame lengths 7, 14 and 28 when $\delta_{offset} = 5ms$, OpenFlow status is checked at $\Delta/3$, and there is no load on the controller.

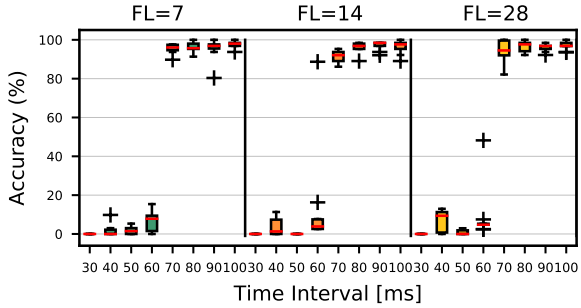


Figure 6: Channel accuracy for time intervals 30-100 ms, and frame lengths 7, 14 and 28 when $\delta_{offset} = 5ms$, OpenFlow status is checked at $\Delta/2$, and there is load on the controller.

our baseline value of $\Delta/2$. When we check the status at $\Delta/3$, we observe a negative influence on the channel, i.e., time intervals 50-70 ms are not effective. In particular, we note that the 70 ms time interval is the operational edge when δ_{delay} is at $\Delta/3$. The reason for these marked changes is the following: The time at which the receiver checks the OpenFlow connection status is crucial. Done too soon, it is likely to detect a zero, and done too late, it is likely to detect a one.

Based on our design, detecting a 1 as a 0 reduces the accuracy more than detecting a 0 as a 1: missing the *SoF* bit (1) can lead to missing the entire frame, and detecting zeros for all the data bits results in the *EoM*. Combining the two can drastically bring down the accuracy which is evidenced when we check the status at $\Delta/3$.

Effect of Message Length $|M|$: To ensure that our channel can sustain longer messages, we measured the accuracy of sending 512 and 1024 byte messages with and without load. The accuracy in each case was very close to the 64 byte message, hence we chose not to show the results here.

Effect of Load on the Controller: Having determined time intervals, frame lengths and δ_{delay} values with close to 100 % accuracy, we compare them with measurements when the controller is under load, as real OpenFlow network can operate with more than two switches. Fig. 6 and 7 illustrate the results from this experiment.

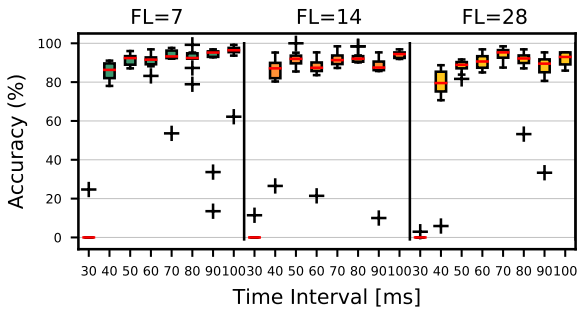


Figure 7: Channel accuracy for time intervals 30-100 ms, and frame lengths 7, 14 and 28 when $\delta_{offset} = 5ms$, OpenFlow status is checked at $2\Delta/3$, and there is load on the controller.

Naturally, load on the controller reduces the accuracy of our channel. Other switches trigger events at the controller which introduces queuing and processing delays for the sender's and receiver's messages. This introduces errors for time intervals that were previously highly accurate, e.g., 60 ms and checking the OpenFlow connection at $\Delta/2$ (Fig. 6) drops to roughly 10% when the controller is under load. Although there is a drop in the accuracy when we check the connection at $2\Delta/3$ (Fig. 7), the smaller time intervals, e.g., 50 ms can still operate at or above 90% accuracy.

6. Discussion

Our evaluation demonstrated that switch identification teleportation can be a highly accurate channel for low throughput covert communication in our setup. We also showed that it depends on several factors, e.g., Δ, δ_{delay}, and the system and network conditions. Nonetheless, techniques to detect teleportation in general, and a covert timing channel such as the one presented in this paper are crucial for networks with high security demands. Hence, we briefly discuss detection possibilities. We also describe some limitations and possible improvements for our design and implementation.

Detection and Mitigation: To the best of our knowledge, firewalls and intrusion detection systems do not monitor the OpenFlow sessions. Even if they are, detecting teleportation attacks are non-trivial as they follow the *normal* pattern of (encrypted) OpenFlow sessions. Preventing switch identification teleportation is exacerbated by the fundamental requirement that switches need to uniquely identify themselves to the controller, and that the controller must allow only a single DPID in the network. However, the attack can be deterred if OpenFlow connections are secured via the following hardened authentication scheme: unique TLS certificates for switches, white-list of switch DPIDs at controllers [26] which also includes the switches' respective public-key certificate identifier, and lastly a controller mechanism that verifies the DPID announced in the OpenFlow handshake is over the TLS connection with the associated (DPID) certificate.

Limitations and Improvements: Indeed, our prototype implementation achieves throughput rates in the order of tens of bits per second. However, it is reasonable to assume that the throughput can be increased by, implementing our algorithms in OvS which is programmed in 'C', or using another controller. Consequently, the delays, e.g., δ_{sc}, will be reduced as the response time to events will be faster, e.g., we will not have to rely on $vsctl$ and $ovsdb$ to set/delete the controller. A novel approach to increase the throughput which we have not measured is for the sender and receiver to initiate several concurrent connections to the controller using unique DPIDs for each connection. In this manner, the sender can send as many bits as connections are made, thereby increasing the throughput by the number of connections. Our channel also comes with some system and network level limitations that are difficult to overcome, e.g., time to establish a TCP connection, packet loss along the path to the controller, etc. Furthermore, our design is for uni-directional communication and does not include error correction. A channel from the receiver back to the sender where the receiver acknowledges, e.g., every frame, can boost the accuracy of the channel.

7. Conclusions

In this paper, we described the design, implementation and evaluation of a novel covert timing channel based on the switch identification teleportation technique. Our prototype implementation of our design can achieve throughput rates of up-to 20 bits per second, with an accuracy of approximately 90% even when there is load at the controller. This means that a 2048 byte RSA private key file can be transferred in nearly 13 minutes. Although our *proof-of-concept* implementation is a low bandwidth channel, we discussed techniques to increase the throughput.

Software-defined networks have become the standard way of doing networking in large data centers, and service provider networks are also moving towards such an architecture and paradigm. With Advanced Persistent Threats (APTs) becoming an increasing problem, covert channels such as the one described in this paper become more relevant, e.g., private keys bought in the black market are used for phishing and malware campaigns. Hence, we must design and develop mechanisms to detect and prevent teleportation attacks that gives APTs a way to covertly communicate or exfiltrate data to a command and control center.

Acknowledgments

We thank the anonymous reviewers for their feedback, Prof. Jean-Pierre Seifert and Dominik C. Maier from TU Berlin for their constructive inputs, Brian O'Connor, Kurt Seifried, and the OpenFlow and ONOS security teams. Research (partially) supported by the Danish Villum project *ReNet* and BMBF Grant KIS1DSD032 (Project Enzevalos).

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *IEEE Computer*, vol. 38, no. 4, pp. 34–41, April 2005.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, December 2013.

[3] D. Firestone, "Vfp: A virtual switch platform for host sdn in the public cloud." in *Proc. NSDI*, 2017, pp. 315–328.

[4] B. Mitchell, "Pentagon considering push to software-defined networking," https://www.fedscoop.com/pentagon-considering-push-software-defined-networking/, 2017, accessed: 02-01-2018.

[5] Netronome, "Agilio CX 2x10GbE SmartNIC SMARTNIC FOR HIGH-PERFORMANCE CLOUD, SDN AND NFV NETWORKING," Netronome, Tech. Rep., 2017.

[6] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. NDSS*, 2015.

[7] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *Proc. NDSS*, 2015.

[8] S. Jero *et al.*, "Identifier binding attacks and defenses in software-defined networks," in *Proc. Usenix Security Symp.*, 2017.

[9] K. Thimmaraju *et al.*, "Taking control of sdn-based cloud systems via the data plane," in *Proc. ACM Symposium on Software Defined Networking Research (SOSR)*, 2018.

[10] K. Thimmaraju, L. Schiff, and S. Schmid, "Outsmarting network security with sdn teleportation," in *Proc. IEEE European Security & Privacy (S&P)*, 2017.

[11] Y. Hu, X. Li, and X. Mountrouidou, "Improving covert storage channel analysis with sdn and experimentation on geni," *National Cyber Summit*, vol. 16, pp. 7–9, 2016.

[12] G. J. Simmons, "A secure subliminal channel (?)," in *Advances in Cryptology*, 1986, pp. 33–41.

[13] C. G. Girling, "Covert channels in lan's," *IEEE Trans. Software Engineering*, vol. 13, no. 2, p. 292, 1987.

[14] T. G. Handel and M. T. Sandford, "Hiding data in the osi network model," in *Proc. Intl. Workshop on Information Hiding*. Springer, 1996, pp. 23–38.

[15] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: design and detection," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2004, pp. 178–187.

[16] R. Tahir *et al.*, "Sneak-peek: High speed covert channels in data center networks," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[17] "Snowden: The NSA planted backdoors in Cisco products," 2014, accessed: 02-01-2018. [Online]. Available: http://www.infoworld.com/article/2608141/internet-privacy/snowden--the-nsa-planted\\-backdoors-in-cisco-products.html

[18] M. A. Bishop, *The art and science of computer security*. Addison-Wesley, 2002.

[19] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[20] *OpenFlow Switch Specification*, Open Networking Foundation, 2013, version 1.3.2 Wire Protocol 0x04.

[21] D. L. Mills, "On the accuracy and stablility of clocks synchronized by the network time protocol in the internet system," *ACM Computer Communication Review (CCR)*, vol. 20, no. 1, pp. 65–75, 1989.

[22] Open vSwitch, "Open vswitch," http://openvswitch.org, 2018, accessed: 02-01-2018.

[23] "ONOS wiki home," https://wiki.onosproject.org/display/ONOS/Wiki+Home, 2017, accessed: 02-01-2018.

[24] M. Gilleland and Merriam Park Software, "Levenshtein Distance, in Three Flavors," https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm, 2017, accessed: 02-01-2018.

[25] M. Jarschel *et al.*, "Ofcprobe: A platform-independent tool for openflow controller analysis," in *Proc. IEEE International Conference on Communications and Electronics*. IEEE, 2014, pp. 182–187.

[26] N. Gray, T. Zinner, and P. Tran-Gia, "Enhancing sdn security by device fingerprinting," in *In Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2017.

Blockage-Robust 5G mm-Wave Access Network Planning

Mohammad Nourifar, Francesco Devoti, Ilario Filippini
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
mohammad.nourifar@mail.polimi.it,{francesco.devoti, ilario.filippini}@polimi.it

***Abstract*—Mm-wave technologies are a promising solution to provide ultra-high capacity in 5G wireless access networks. However, the potential of several-GHz bandwidths must coexist with a harsh propagation environment. While high attenuations can be compensated by advanced antenna systems, the severe obstacle blockage effect can only be mitigated by more sophisticated network management.**

One of the most widely adopted techniques to guarantee a reliable service in mm-wave access scenarios is to establish multiple connections from mobile to different base stations. However, the advantage of multiple mm-wave connections can be fully exploited only if uncorrelated channels are available, thus spatial diversity must be ensured. Although smart base-station selections could be made once the network is deployed, much better results are achievable if diversity-aware selection aspects are already included in the network planning phase.

In this paper, we propose for the first time an mm-wave access network planning framework which considers both spatial diversity among potential base-station selection candidates and user achievable throughput, according to channel conditions and network congestion. The results show that our approach allows to obtain much better spatial diversity conditions than traditional k-coverage approaches, and this can indeed provide higher robustness in presence of sudden obstacles.

I. Introduction

In recent years, millimeter-wave (mm-wave) technologies have attracted a lot of interest as one of the main solutions to deliver the multi-gigabit-per-second promise to wireless broadband access users. Although scientific and industrial research has been focusing on the fundamental task of improving the spectral efficiency of wireless links and increasing the network density by deploying more devices, unlocking new spectrum bands looks to be the answer to the need of a radical boost in the achievable throughputs, as required in 5G networks.

The mm-wave spectrum band, only partially occupied, can potentially accommodate several GHz of bandwidth for wireless access communications. However, this opportunity brings in several technical challenges caused by the harsh propagation environment at very high frequencies. The first challenge is to overcome the strong attenuation over distance. This has been tackled by the design of advanced antenna arrays systems, which can concentrate many elements in small form factors thanks to the short wavelength [1]. They allow to typically reach a coverage of several hundreds of meters. A further issue with mm-waves is related to their high penetration loss and limited diffraction [2], which make every obstacle actually opaque, and thus a cause of link blockage. In order to break this limit, we must resort to new network architectures and smart resource management algorithms.

ISBN 978-3-903176-08-9

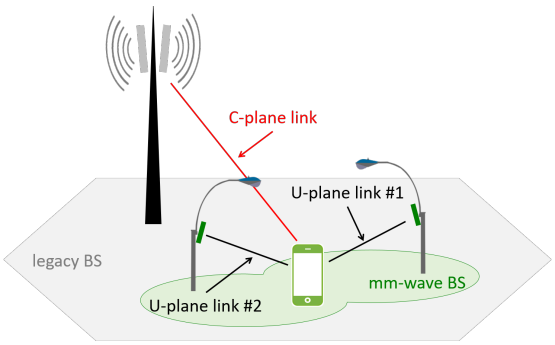


Fig. 1: Example of multi-connectivity architecture

Although directional antennas allow to span several hundreds of meters, it was immediately clear that the severe blockage effects prevented considering a wide access network entirely based on mm-wave technologies. Indeed, heterogeneous network architectures have been proposed to guarantee a full coverage with a separated and reliable control-plane delegated to legacy macro base stations, while a multi-gigabit user plane is provided by the on-demand activation of mm-wave small-cells in specific regions of the service area [3]–[5]. Each device can potentially establish a dual inter-technology connection according to the current user, network, and application contextual information. However, the ultra-high-speed mm-wave service is still offered in an opportunistic manner, as such an inter-technology dual-connectivity only guarantees a reliable network signaling.

Intra-technology connectivity to multiple sites equipped with the same radio interfaces has been used for many years to facilitate multi-user and hand-over procedures [6]. In the last few years, it has emerged as a solution to improve the user throughput in LTE cellular systems via inter-site carrier aggregation [7]. Finally, multi-connectivity as a solution to the unreliability problem of the ultra-high-speed service has been explicitly envisioned in 5G systems, particularly when dealing with mm-wave access networks [4], [8], [9], an example of mm-wave-technology multi-connectivity is provided in Fig. 1.

Intra-technology multi-connectivity clearly implicates higher complexity, however, this is definitively counterbalanced by many advantages for mm-wave communications: i.e., i) in a scenario with very unstable links, more alternatives ensure that the user can stay connected

to at least one mm-wave base station (BS) when obstacles suddenly appear, ii) the establishment of multiple connections avoid a full directional cell discovery phase [3] in case of a single connection drop, iii) multiple simultaneous connections allow to apply refinement techniques to precisely localize a user and improve the efficiency of context-aware resource allocation algorithms. However, in order to be effective, intra-technology multi-connectivity requires selected mm-wave BSs to experiment uncorrelated channel conditions. Therefore, we need spatial diversity among selected mm-wave connections [10].

Since mm-wave transmissions are highly directional, ensuring an angular separation between two mm-wave BSs with respect to every single user is a good way to provide spatial diversity. Clearly, the larger angular separation between two BSs, the higher channel uncorrelation between their channels, thus better spatial diversity can be achieved. Smart spatially-diverse BS selections can be made relying on the available network layout. However, we believe (supported by the obtained results) that much better angular separation can be achieved if mm-wave BS locations are optimized by a planning process that includes spatial diversity aspects. This cannot be done by traditional k-coverage approaches, as they ensure k-connectivity without considering spatially diversity, which is potentially further reduced by the goal of minimizing the number of installed BSs.

In this paper, we propose a new network planning approach to provide intra-technology multi-connectivity in mm-wave access networks by maximizing the angular separation, thus the spatial diversity, in the group of BSs each user can select. In order to provide a realistic network plan, which includes capacity aspects, we also consider the achievable user throughput, determined by both user-BS channel conditions and congestion at the selected BSs. Finally, our approach allows to dimension a residual capacity in each BS to serve as a backup for user requests interrupted by unexpected obstacle blockages.

To the best of our knowledge, this article considers, for the first time, spatial-diversity in the design of mm-wave access networks to provide robustness in front of sudden blockages. In addition, we propose a viable approach that can address realistically sized instances and provide a thorough evaluation of the advantages with respect to traditional network planning approaches based on k-coverage.

The remainder of the paper proceeds as follows. In Section II, we describe the aforementioned problem and provide an overview of our solution. Section III includes the mathematical programming models used in our approach, while in Section IV we show and comment the results of the numerical evaluation we performed. In Section V, we discuss the state-of-the-art on mm-wave access network planning and directional communications. Then, Section VI concludes the paper with some final remarks.

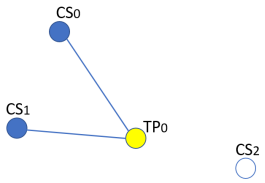
II. The mm-wave Network Planning Problem

Network planning is a key phase of the network deployment process, which determines base-stations' placement and configuration while considering a number of aspects, including signal propagation, traffic distribution, interference, deployment cost, etc. In this process, a discrete set of candidate sites (CSs) describes the possible sites suitable for BS placement. The traffic distribution is represented using a discrete set of points, test points (TP), which are considered as centroids of traffic. The adoption of mm-wave technologies in the mobile radio access network implies a radical change in propagation conditions and antenna technologies, which lead to specific service features that must be taken into account to effectively plan the network. To be more specific, mm-wave communications are characterized by strong path losses and are vulnerable to strong fading, resulting in high signal-blockage probability, which if not properly addressed, leads to an unreliable service.

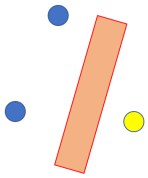
Real environments are characterized by the presence of objects (e.g. trees, vehicles, human bodies, etc.) that can lie between transmitters and receivers. Due to the high frequency characterizing mm-wave communications, almost every single object is opaque to mm-wave propagation, leading to a high probability of path obstruction that causes severe signal attenuation and connectivity drop. Therefore, the presence of objects should be carefully considered in the network planning phase to enable the deployment of a reliable mm-wave access service able to satisfy 5G QoS requirements.

As far as mm-wave radio planning is concerned, we can distinguish two different categories of objects: static objects, (e.g. buildings, walls, etc.), and nomadic objects (e.g. cars, trucks, pedestrians, etc.). While the deterministic nature of static objects' position allows to easily take into account their presence by means of propagation prediction tools, nomadic objects can cause unpredictable connectivity drops, therefore, they can strongly worsen ultra-high-speed reliability. In this perspective, while static obstacles do not substantially change the way in which radio network planning has been carried out so far, the need of a reliable mm-wave service requires to plan the network in such a way that a potential user can be reached by multiple mm-wave BSs, providing users with backup links to restore their connectivity in case of blockage caused by nomadic obstacles. Effectively providing backup links, however, does not translate in a mere densification of the base-stations' placement as their locations can strongly impact on the final outcome, and if not properly managed, can make additional base stations useless.

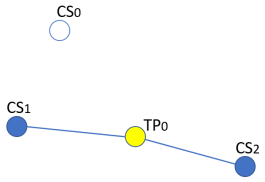
Fig. 2 shows a simple example by comparing two possible multi-coverage solutions, where two BSs must be installed in CSs covering TP_0. The two solutions are represented in Fig. 2a and 2c and both are valid solutions of the problem, however, the two are not equivalent in terms of robustness to obstacle obstructions. Fig. 2b and 2d show the different behavior of the two solutions in case an unexpected obstacle appears. In Fig. 2b the random obstacle is completely obstructing all possible TP connection alternatives. In terms of robustness, this 2-coverage solution has almost the same quality as that of a single coverage, causing the additional base station to be useless. Instead, in Fig. 2d, thanks to the angular separation of installed base stations, the TP can still maintain an mm-wave



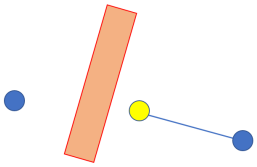
(a) Solution 1



(b) Obstacle blockage in Solution 1

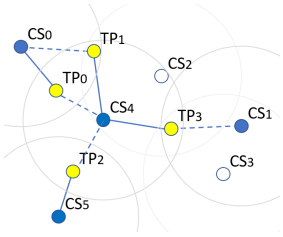


(c) Solution 2

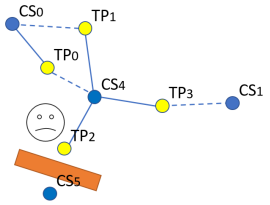


(d) Obstacle blockage in Solution 2

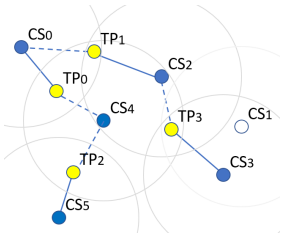
Fig. 2: Obstruction examples in two different 2-coverage solutions



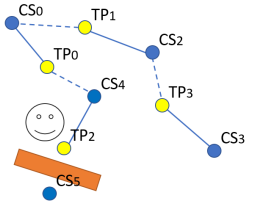
(a) Solution 1 w/o capacity reservation



(b) Obstacle blockage in Solution 1



(c) Solution 2 w/ capacity reservation



(d) Obstacle blockage in Solution 2

Fig. 3: Obstruction examples with different capacity reservation strategies

connection.

In this paper, we propose a network planning approach based on mathematical programming models which can effectively improve the mm-wave service robustness against sudden obstacle blockages. The rationale behind our approach is to include spatial-diversity aspects in a k-coverage problem in order to provide the required coverage while maximizing the angular diversity from which multiple mm-wave BSs can reach a potential user. The angular diversity will increase the availability of independent backup connections in case of obstacles obstruction, and consequently, it will improve the mm-wave service reliability.

However, describing the problem just in geometrical terms by considering the physical availability of backup connections only ensures the mm-wave service coverage, but does not provide any guarantee on its quality. Indeed, radio resources are shared among users associated with a BS. Each user occupies a portion of radio resource according to its demand, achievable modulation scheme, and network congestion. Therefore, mm-wave BSs must provide enough throughput even in case of link reconfigurations due to obstacle obstructions. This must be reflected into the network planning process. To this extent, we assume each user has a minimum traffic demand to be guaranteed and it must be entirely served via one link (primary link) of the multiple connections made available in the network plan. Other links (secondary links) are kept synchronized, but traffic is sent only in case the primary link is blocked. This is a simplifying assumption, although in line with current technology advancements, which allows to better understand the trade-offs involved in this problem. However, other solutions, like coordinated multipoint or cooperative transmissions, can be easily captured in the proposed models by simply making straightforward changes.

There are two opposite approaches to deal with demand guarantees in case of link reconfigurations in this mm-wave scenario. The most conservative solution is to plan the network in such a way that in each of the alternative connections (one primary link and several secondary links) the required demand is guaranteed. This ensures that the request is satisfied for any obstacle obstruction that does not completely block all possible connections. However, this implies high installation costs due to the resource underutilization when obstacles impairments are not severe. The opposite solution consists in just guaranteeing the demand through the primary link, with no reservation on the others. This provides the minimum cost deployment, however users may see a reduction of guaranteed throughput in case of link reconfiguration. Clearly, an intermediate behavior would provide the best trade-off: the whole demand can be reserved on primary links, while only a fraction of it along secondary links. This allows to mitigate the effects of the reconfigurations leveraging link failure statistics.

Fig. 3 shows an example of two possible solutions applying the two opposite capacity reservation strategies. In the example, the network planning has to provide 2-coverage (one primary and one secondary link, respectively solid and dashed lines) to TPs, and BS capacity, C, is such that only two user demands, D, can be accommodated with no loss, i.e., $C = 2D$. The figure shows a network snapshot: Fig. 3a is the solution without capacity reservation on the secondary link, while Fig. 3c with full capacity reservation. Fig. 3b and 3d show the effects of a random blockage in both solutions: despite being equivalent in terms of connectivity, they are not in terms of achievable throughput. Indeed, without capacity reservation, Fig. 3b, TP_2 secondary link is not a good backup because CS_4 resources are already completely saturated by TP_1 and TP_3. Vice-versa, when full capacity reservation is guaranteed, Fig. 3d, CS_4 resources are totally reserved to TP_0 and TP_2 secondary links, therefore the link reconfiguration caused by the obstacle does not impact on TP_2 guaranteed throughput.

In the next section, we present two mathematical programming models able to capture all above-mentioned aspects and provide an obstacle-robust mm-wave network plan.

III. Network Planning Models

This section describes the Mixed-Integer Linear Programming (MILP) models we propose for blockage-robust 5G mm-wave planning. We firstly present a basic optimization model to include angular diversity aspects in multiple coverage, with the aim of maximizing angular diversity among BSs selected for the coverage. Then, we propose an extension of the basic model to jointly plan coverage and guarantee expected throughput. We include both the extra capacity needed to manage backup connections in case of blockage and the effect of rate adaptation techniques.

A. Maximizing Angular Diversity

Considering an area to be covered by an mm-wave service, we denote by $\mathcal{M}$ the set of CSs where a BS can be installed and by $\mathcal{N}$ the set of TPs. The objective of the proposed model is the maximization of the angular diversity from which each TP connects to the BSs selected for multi-connectivity, while satisfying k-coverage and deployment cost constraints.

We start by introducing parameters and decision variables used in our model to provide blockage-robust coverage. The coverage matrix $\mathbf{A} = A_{i,j}$ summarizes propagation characteristics in our model. $A_{i,j}$ depends on physical properties, like distance between TP i and CS j, transmitting power, receiver sensitivity, and antenna gain, and it is equal to 1 if the CS j can cover the TP i (when they reciprocally point their beams) and 0 otherwise. These coverage maps are commonly adopted in any radio network planning approach. Moreover, this is flexible to any assumption on physical properties: a proper matrix will be filled, and thus the model applied. Note that even static obstacles can be considered in this formulation. Indeed, the presence of a fixed obstruction will translate into a set of 0s at specific (i, j) pairs, which would have been 1s otherwise. Finally, we accounted for highly directional antennas by averaging the directivity function over the main lobe in order to obtain a realistic value of the antenna gains even in case of non-perfect transmitter-receiver alignment.

Angular diversity is evaluated through the matrix $\mathbf{\Theta} = \Theta_{i,j,k}$, which denotes the angular separation between two different CSs $j, k \in \mathcal{M}$, observed from the point of view of a TP $i \in \mathcal{N}$. This matrix can be automatically computed a-priori, once TP locations and potential BS sites (CSs) are known. Parameter K defines the minimum coverage level (K-coverage), that is, the minimum number of installed mm-wave BSs to cover each TP in a valid network plan. Parameter B denotes the deployment budget limiting the number of activated CSs.

The model considers two main types of decision variables:

- A binary installation variable y_j, which defines the mm-wave BS placement within available CSs, y_j is equal to 1 if a BS must be installed in CS j, 0 otherwise.
- A binary association variable $x_{i,j}$, which defines TP-CS assignment. In the optimal solutions, $x_{i,j} = 1$ means that CS j is selected as one of the K alternative links for TP i which provide the best BS angular separation.

TABLE I: Decision variables, set and parameters used in the models

SETS	
$\mathcal{N}$	Set of TPs
$\mathcal{M}$	Set of CSs
PARAMETERS	
B	Deployment budget
K	Minimum coverage
Θ_{ijk}	Angular separation between CSs j and k seen from TP i
A_{ij}	Coverage between TP i and CS j
S_j	Installation cost of CS j
D_i	Demand of TP i
C_j	Capacity of CS j
R_{ij}	Max rate between TP i and CS j
VARIABLES	
x_{ij}	Assignment between TP i and CS j
y_j	Installation of CS j
p_{ij}	Definition of link between TP i and CS j as primary link
δ_i	Minimum BS angular diversity seen by TP i

The additional variable $\delta_i \in [0, 2\pi]$ is a support variable denoting the minimum angular diversity achievable by TP i. In order to simplify the description of the following models, the definition of their variables and parameters are summarized in Tab. I.

Given the above definitions and notation, we can describe the *Angular-Diversity-aware k-coverage Problem* (**ADkP**) as follows:

$$[\mathbf{ADkP}]: \quad \max \quad \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \delta_i \tag{1}$$

s.t.

$$\delta_i \leq \Theta_{ijk} + 2\pi * (2 - x_{ij} - x_{ik}), \qquad \forall i \in \mathcal{N}, \forall j, k \in \mathcal{M} : j \neq k \tag{2}$$

$$\sum_{j \in \mathcal{M}} x_{ij} \geq K, \qquad \forall i \in \mathcal{N} \tag{3}$$

$$x_{ij} \leq A_{ij} \cdot y_j, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{4}$$

$$\sum_{j \in \mathcal{M}} S_j y_j \leq B, \qquad \forall j \in \mathcal{M} \tag{5}$$

$$x_{ij}, y_j \in \{0, 1\}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{6}$$

The objective function (1) maximizes the average of the minimum angular diversity values achievable at each TP $i \in \mathcal{N}$. [1]

Constraint (2) is the key constraint for providing angular diversity to the k-coverage framework. It holds only if CS j and CS k are selected as CSs providing the best angular separation to TP i in the optimal solutions (thus $x_{ij} = x_{jk} = 1$),

[1] Note that different objectives can be easily plugged into the model, we selected this function as it allows to achieve a good balance between diversity fairness and overall diversity maximization.

otherwise the constraint is inactivated via a big-M technique. If CS j and CS k are selected and assigned to TP i, then their angular separation Θ_{ijk} must be taken into account. This must be true also when k-coverage has $k > 2$. In this case, we must evaluate the angular separation of every CS pair in the set of selected CSs. The combination of objective function and constraint (2) acts like a max-min function, which forces variables δ_i to assume a value equal to the minimum angular separation between every possible pair of CSs among those selected to provide the best angular diversity to TP i. Constraint (3) is the coverage constraint and ensures that the required coverage level K is met per TP. Constraints (4) and (5) respectively enforce that a TP can be assigned only to a covering CS with a BS installed and that the installation cost (S_i is the cost of installing an mm-wave BS at CS i, including backhauling costs) does not exceed a given budget B.

B. Advanced models considering user throughput

We now extend *ADkP* model to consider capacity planning as well. In this scenario, the network plan must guarantee, together with the desired coverage level, that the user traffic demand is met[2]. Therefore, we include in our model the user throughput demand associated with TP i through parameter D_i, and, through parameter C_j, the capacity associated with the installation (backhauling included) of an mm-wave BS in CS j.

We also introduce a further set of binary decision variables p_{ij}. Variable p_{ij} is set to 1 if the link to CS j is selected to be the primary link for TP i, i.e., the preferred link to convey user traffic, the one with a full throughput guarantee. We refer to other $K-1$ links as secondary links. We would like to remark that the model does not mandatorily imply any capacity reservation mechanism. Considering a demand guarantee in the network planning phase has just the goal of providing a network configuration in which the potential throughput available to each user can be above a given threshold.

Finally, we use parameter $\alpha \in [0,1]$ to tune extra-capacity reservation in order to deal with link reconfiguration. With $\alpha = 1$, throughput is guaranteed over all k alternative links, reserving a full extra-capacity on secondary links. Vice versa, $\alpha = 0$ means that throughput is guaranteed only on primary links (those defined by variables p_{ij}). Values between 0 and 1 provide a plan in which the entire throughput is guaranteed only on primary links, while a fraction α of it is guaranteed on secondary links, limiting the reserved extra-capacity.

The *Joint Angular-Diversity-and-Capacity-aware k-coverage Problem* (**JADCkP**) is described as:

$$[\mathbf{JADCkP}]: \quad \max \quad \frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}}\delta_i$$

s.t.

$$[\mathbf{ADkP}]: \text{ constraints (3)-(5)}$$

$$p_{ij} \le x_{ij}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \quad (7)$$

$$\sum_{j\in\mathcal{M}} p_{ij} = 1, \qquad \forall i \in \mathcal{N} \quad (8)$$

$$\alpha \sum_{\substack{i\in\mathcal{N}:\\ A_{ij}=1}} D_i x_{ij} + (1-\alpha) \sum_{\substack{i\in\mathcal{N}:\\ A_{ij}=1}} D_i p_{ij} \le C_j y_j, \qquad \forall j \in \mathcal{M} \quad (9)$$

$$x_{ij}, y_j, p_j \in \{0,1\}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \quad (10)$$

Three new constraints have been added to the previous model. Constraints (7) and (8) guarantee that only one among CSs assigned to TP i can be defined as primary link. Capacity constraint (9) enables the throughput guarantee strategy defined by the parameter α. Setting $\alpha = 0 (\alpha = 1)$ inactivates the first(second) LHS term. For $\alpha \in (0,1)$, the model enforces that the sum of secondary links' extra capacity and primary link's full demand does not exceed the site capacity C_j, if an mm-wave BS is installed. Otherwise, no demand can be served. All remaining constraints are the same as in the previous model.

Rate Adaptation: JADCkP model can be extended to deal with link rate adaptation, which dynamically selects the proper code and modulation scheme according to the channel quality. In order to capture this behavior, we introduce the matrix $\mathbf{R} = R_{ij}$, which defines for each potential TP i - CS j pair the maximum achievable rate. Matrix $\mathbf{R}$ can be filled considering physical link parameters like transmission power, antenna patterns, propagation conditions, per-modulation receiver sensitivity thresholds, transmission overheads, etc. Moreover, given the very-high directivity of involved transmissions, we can reasonably assume that achievable rates are only slightly affected by concurrent transmissions, thus interference can be modeled as a simple traffic demand overhead.

In order to enable rate adaptation features in *JADCkP* model, we need to replace constraint (9) with the following:

$$\alpha \sum_{\substack{i\in\mathcal{N}:\\ A_{ij}=1}} \frac{D_i}{R_{ij}} x_{ij} + (1-\alpha) \sum_{\substack{i\in\mathcal{N}:\\ A_{ij}=1}} \frac{D_i}{R_{ij}} p_{ij} \le y_j, \qquad \forall j \in \mathcal{M} \quad (11)$$

Differently from (9), in which the simple user bitrate is considered, constraint (11) models mm-wave BS resource sharing as a time-sharing process (similarly to the indications of IEEE 802.11ad frame specification).

The rationale behind constraint (11) is that an average rate for user i equal to the maximum achievable rate R_{ij} can be obtained only if just user i is served for the entire duration of the available time (i.e., time frame). The fraction $\frac{D_i}{R_{ij}}$ expresses the time share at CS j to be assigned to TP i to get an average rate equal to D_i, given a R_{ij} bit/s channel. The constraint enforces that the sum of the time shares of TPs assigned to CS j does not exceed the physical limit of 1, if a mm-wave BS is installed in CS j.

IV. Numerical Results

In this section, we provide the results of a numerical evaluation campaign on previously presented models. All the following instances are modeled and solved by IBM ILOG OPL and CPLEX Optimizer, and, wherever not differently specified, each outcome is the result of an average over 100 different instances. In each instance, we consider a rectangular

[2]The model can equally consider uplink traffic, downlink traffic, or their sum.

service area with dimensions $800m \times 600$m, a number of m candidate sites, in which to locate mm-wave BSs, and a number of n TPs. Using a pseudo-random number generator each CS and each TP is assigned a position with uniform distribution in the service area. Without loss of generality, we assumed that BS deployment cost is the same in all CSs and equal to 1.

Although dealing with a NP-complete problem, as including a set-covering problem as sub-problem, the solution of large instances of 60 TPs and 100 CSs just took 10 minutes with a 2% optimality gap on an Intel Xeon 2.4 GHz and 96GB RAM 8-core machine. Considering the deployment process of broadband wireless access networks, this is a very reasonable time for an entire plan.

We set the CS transmission power at 30 [dBm]. The average antenna gain over all possible main-lobe directions is found by averaging the antenna model provided by [11] with fixed elevation and azimuthal beam-width respectively set at 60 [deg] and 20 [deg], leading to an average gain of 9.45 [dB]. In order to fill the coverage matrix $\mathbf{A}$, we considered the mm-wave propagation model developed within MiWEBA project [11].

When rate adaption has been considered, we set rates and SINR thresholds as those used IEEE 802.11ad (WiGig) specification [12]. While the maximum BS capacity is set to 4.6Gbps, which corresponds to the maximum achievable rate in IEEE 802.11ad.

It is also important to introduce the performance figure Average Angular Diversity (A-AD) in a way that it makes the demonstration and comparison intuitive and easy to understand, therefore we use this simple formulation for A-AD in our following plots:

$$\text{A-AD:} \qquad \frac{f_{avg\_min}}{(2\pi/K)}$$

where f_{avg_min} is the optimum value of the objective function of the proposed models, that is, the average over the minimum angular diversity values achievable at each TP and K refers to the K-coverage parameter. A-AD can vary from 0 to maximum of 1 in the case where the average angle between CSs covering each TP is the maximum possible for all TPs. This is best case scenario as all CSs are well positioned with respect to TPs and there is a high Average Angular Diversity. In the case of traditional k-coverage where the concept of A-AD does not exist, having all the CSs selected by a standard min-cost k-coverage model, we use another MILP model to compute the best possible A-AD with those previously selected CSs.

Fig. 4 shows, with solid lines, the behavior of A-AD when number of available CSs and deployment budget vary. In addition, dotted lines show the percentage of available CSs which are used to install an mm-wave BSs. Given a fixed budget B, A-AD is increasing as the number of CSs increases in the service area. This clearly demonstrates that having more potential candidate sites provides better choices to the network planning. Similarly, as budget B increases, higher A-AD can be achieved. The comparison with the results of a traditional k-coverage approach that minimizes the total number of selected BSs shows two interesting aspects: i) when the number of CSs is limited or the budget is low, the number of deployable BSs is so limited that coverage is the main focus and little can be done in terms of angular diversity; ii) when budget increases, our model can achieve much better A-AD values and better exploit available degrees of freedom. The latter is also confirmed by dotted lines in the picture, which show that the number of installed CSs for $B = 12$ is similar to that of the k-coverage. In addition, differently from the traditional k-coverage, our model allows to exploit high budget values, by substantially increasing the number of installed CSs. The decrease in the number of installed CSs when more CSs are available is due to the considered objective function: when more and better choices can be made (more CSs), the same performance (A-AD) can be obtained with less resources (installed CSs).

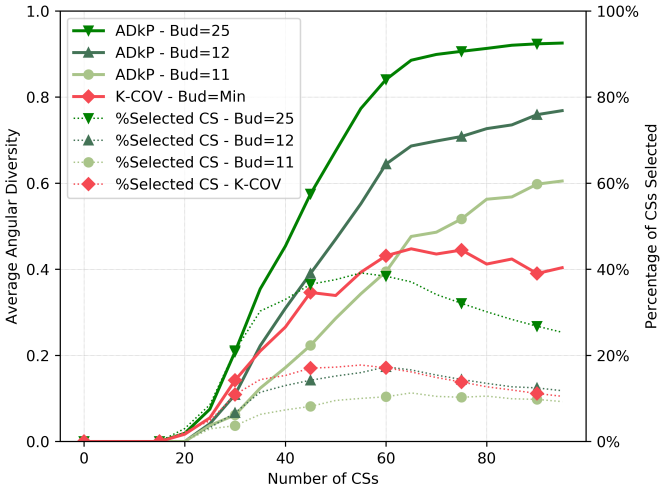


Fig. 4: A-AD behavior varying the number of available CSs and deployment budget in the proposed model (ADkP) and in traditional k-coverage (K-COV). The scenario considers 15 TPs and $K = 2$.

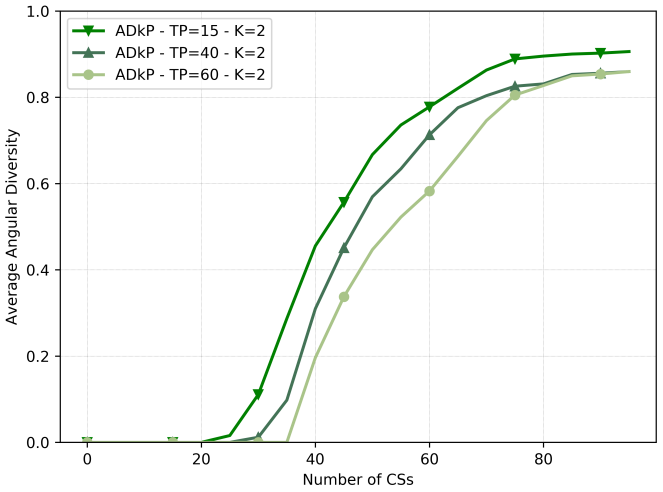


Fig. 5: A-AD behavior varying the number of available CSs and dropped TPs. The scenario considers a budget equal to half of the number of available CSs and $K = 2$.

Fig 5 explores the cases with a higher number of TPs and its impact on A-AD. Increasing the number of TPs in the

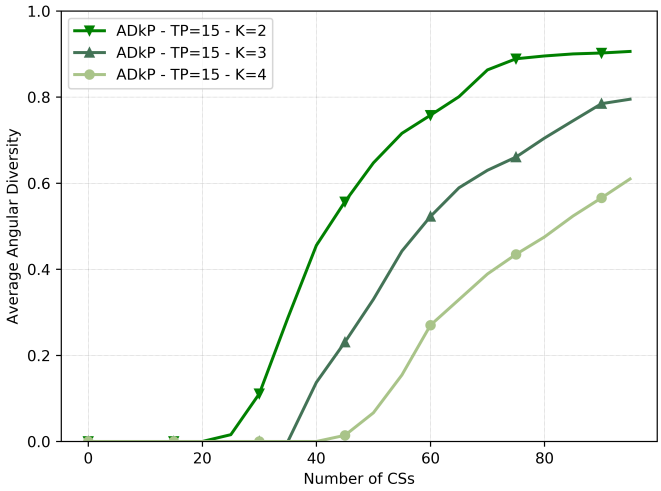


Fig. 6: A-AD behavior varying the number of available CSs and coverage parameter k. The scenario considers a budget equal to half of the number of available CSs and 15 TPs.

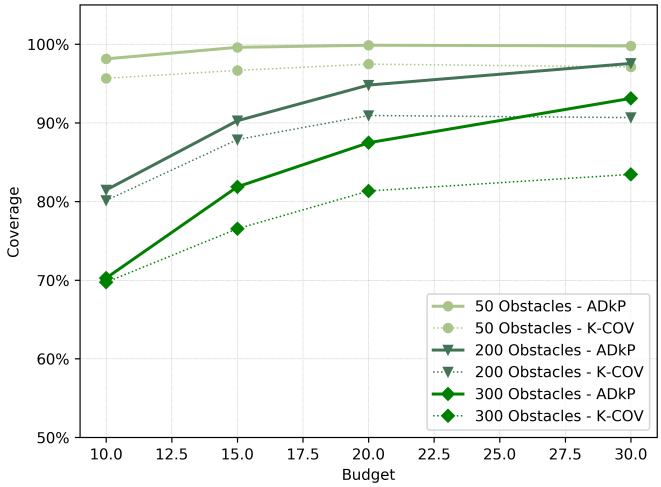


Fig. 7: Coverage Comparison between the proposed model (ADkP) and a classical k-coverage (K-COV) varying the deployment budget and the number of obstacles in the service area. The scenario considers 15 TPs and 100 available CSs.

service area with a fixed budget, a slight decrease in A-AD is observed. This is due to the fact that more TPs are scattered all over the service area and more mm-wave BSs are needed to provide a given angular diversity to each of them.

Besides the impact of higher number of TPs, Fig 6 shows the effect of having higher K as coverage constraint. Higher K means more CSs should cover each TP, which in turn means the coverage constraint (3) will be tighter, therefore, we will see the impact as an increase in the minimum number of needed CSs. Similarly, A-AD will be relatively lower at fixed budgets.

Previous figures describe and summarize the behavior of the proposed model, showing a remarkable advantage in terms of achievable angular diversity with respect to a traditional k-coverage approach. We show now how a better angular diversity translates in higher robustness in front of random obstacle blockages.

To prove the blockage robustness of the proposed mm-wave access network planning approach, we consider some line segments randomly dropped in the service area with a random orientation. Each 20m segment behaves like an obstacle surface by blocking the mm-wave propagation, so that a CS-TP link is interrupted if the link and an obstacle segment intersect. Adding each of these obstacles, we check blockages in both traditional k-coverage and proposed model with fixed budgets, where k-coverage model randomly selects CSs to fill the excess budget.

In Fig 7 we show the network robustness once the network is planned (according to either the proposed model or a k-coverage approach) with different budget values and random obstacles are dropped into the service area. The robustness in front of sudden obstacles is measured as percentage of TPs that can still get connected to a mm-wave BS after the appearance of a given number of obstacles. We clearly notice the coverage and robustness difference between the proposed approach and the traditional approach by increasing the budget. There are three important points to be considered:

1) In the worst case scenario, when budget B is very low (in our case 10), as it is clear also on Fig 4, there is no much degree of freedom to increase A-AD, as a result, the proposed model and the k-coverage mostly select similar CSs with some minor changes. The reason is budget B is so tight that our proposed model has no way but a solution very close to the k-coverage case, so in cases with low budget, a small increase in the coverage is observed.
2) As we provide our model with more budget, we have a higher chance to increase the amount of A-AD, which gives the opportunity to improve the total coverage after the obstacles drop. The difference in coverage is remarkable in Fig 7 with 200-300 obstacles in the service area.
3) The other important message that Fig 7 conveys is that it is not true that by haphazardly adding the excess CSs we can get the same coverage as by positioning them with high A-AD using the proposed model. The increase in A-AD will always result in lower blockages caused by obstacles.

This summarizes the general behavior of the proposed planning model by mentioning all trade-offs we have, and finally proves its advantages with respect to traditional k-coverage.

In the following part, we investigate the performance of the advanced model, which jointly considers coverage, capacity reservation and user throughput via rate adaptation. Fig. 8 shows a typical effect caused by the addition of user demand and rate adaptation features: as the demand increases, the average distance between CSs and their assigned TPs decreases. Since we need higher data rates for each TP, they can be delivered only in good channels conditions, like those nearby mm-wave BSs. By increasing user demands, A-AD decreases as well, as it can be seen in Fig 9. Indeed, requesting high data rate results in a lower number of CSs close enough to provide that throughput, so the model has less freedom to position mm-wave BSs and A-AD decreases, as a result.

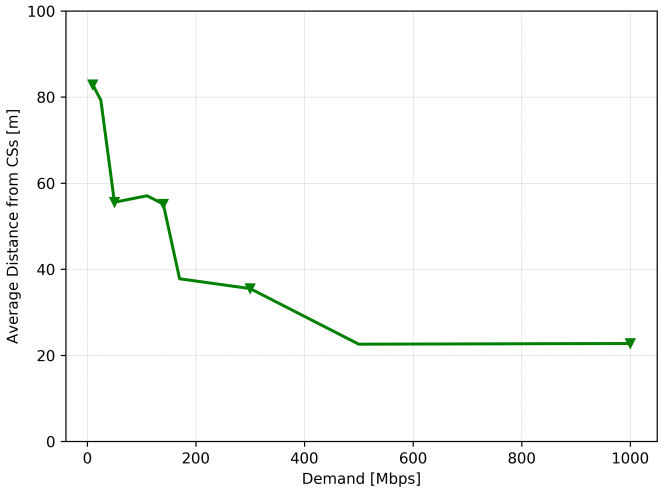


Fig. 8: Average primary link length varying per-user demand. The scenario considers 15 TPs, 200 CSs, $B = 30$, and $\alpha = 0.5$.

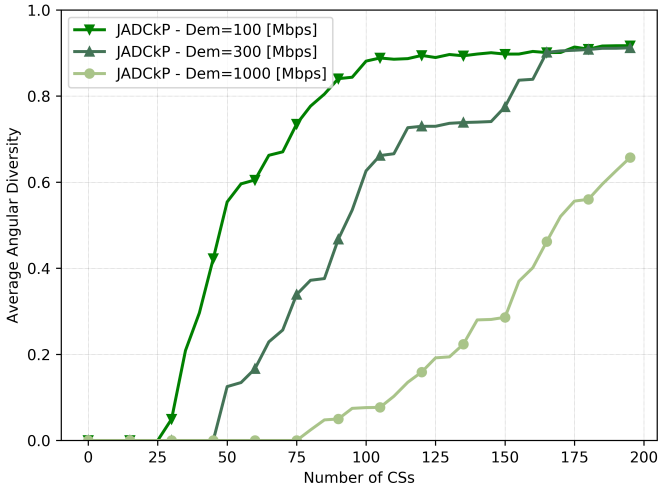


Fig. 9: A-AD trend varying the number of available CSs and user demands. The scenario considers 15 TPs, 200 CSs, B is equal to half of the number of available CSs, and $\alpha = 0.5$.

One feature which plays an important role in this model is α, which can increase the amount of extra capacity reserved on secondary links. As we add obstacles into the service area, the average throughput is plotted in Fig. 10 for different values of budget and α. Reported values are obtained by solving a throughput-maximization assignment model over the links that are still available once obstacles are dropped. The figure clearly shows the effects of modifying parameter α: in the case of $\alpha = 1$, the average throughput is higher as we have higher reserved capacity in our secondary (backup) links. Moreover, high-budget values reduce the difference between cases with $A = 0$ and $A = 1$, as the total network capacity is much higher than the total demand in the service area. This, together with a network plan with high A-AD, helps to provide higher average throughput in case of sudden blockages, even if no capacity reservation is prearranged.

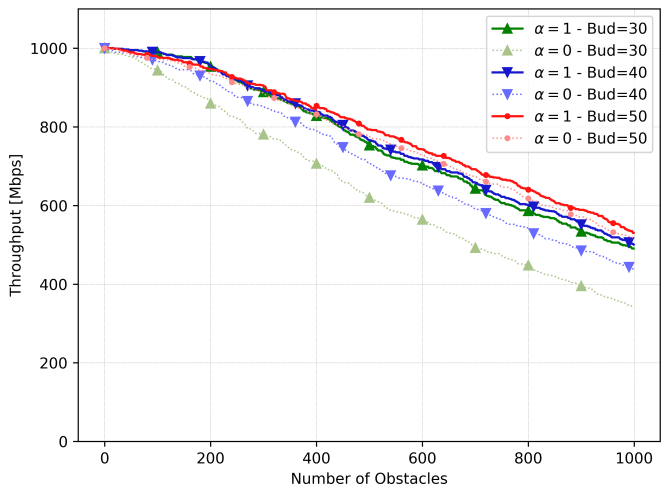


Fig. 10: Available throughput with JADCkP when obstacles appear. Comparison for different values of α and budget. The scenario considers 15 TPs, 200 CSs, and $D_i = 1000$ Mbps.

V. Related Works

Within the 30-year-old literature on wireless network planning, directional transmissions have been addressed in a big set of works dealing with different network topologies. In wireless ad-hoc networks, the availability of directional antennas increases the degrees of freedom in which the common medium can be shared; advanced medium access [13], scheduling [14], and topology control [15] problems have been investigated. Models for the joint optimization of routing and transmission scheduling in wireless mesh networks [16] have been proposed to fully exploit the potential of directional transmissions in order to improve the capacity of those networks. Finally, directional sensors have been considered in models for planning Wireless Sensor Networks (WSNs) [17]. However, despite inspiring 5G network approaches, these models do not capture all specific aspects of mm-wave communications.

In the context of Wireless Personal Area Networks (WPANs), mm-wave directional transmissions were introduced more than 10 years ago. This has led to several optimization papers investigating many different aspects related to the use of mm-waves for transmissions among mobile users: spatial multiplexing exploitation via transmission scheduling [18], optimal admission control for domestic high-definition video streams [19], best relay identification in multi-hop communications [20]. Unfortunately, WPAN scenarios are radically different from those characterizing 5G mm-wave access networks. The very short WPAN range makes the use of omnidirectional transmission still viable in some communication phases and reduces the probability of LOS obstacle blockage. In addition, WPANs are typically designed according to traffic requirements very different from those of mm-wave 5G networks. Therefore, we need new models and methods to deal with the specific aspects of such networks.

When mm-wave technologies are involved, obstacles' shadowing and blockage effects become one of the major issues during network operations: access [3], resource management

[21], [22], transport layer [23], etc. Different solutions have been proposed to mitigate these effects, multi-connectivity is the most common in the mm-wave context, where the management of these multiple connections is the main focus [4], [8], [9].

The standard way to guarantee multi(k)-connectivity in a wireless access network is to adopt planning methods with k-coverage constraints. The literature on k-coverage, mostly for WSNs [24], is huge, indeed many objectives and characteristics can be requested to the obtained network layout. In the field of Visual Sensor Networks, the problem of how to place cameras in order to avoid obstacles has been largely studied [25]. The goal is to plan their fields of view in order to provide the best visibility of a given area. Although sharing some similarities with our problem, different technological domain and lack of throughput constraints make these approaches unsuitable for our purposes.

To the best of our knowledge, this is the first paper that investigates angular diversity for connection reliability and capacity aspects in a wireless multi-connectivity context.

VI. Conclusion

In this paper, we have proposed a network planning approach for 5G mm-wave access networks that allows to fully exploit multi-connectivity by providing spatial diversity among BSs. This produces network layouts with better BS alternatives for mm-wave users. In addition, QoS aspects related to user throughput guarantees and rate adaptation have been included as well.

Our approach has been tested on different instances, showing it is indeed effective in providing an angular separation between BSs remarkably larger than that achievable with traditional k-coverage approaches. This leads to networks that are much more robust to unexpected obstacles. In addition, results have shown that capacity reservation strategies and rate adaptation play a main role in defining the final network design.

We believe multi-connectivity will be a fundamental feature of 5G mm-wave networks and our contribution can help to provide the required level of reliability to such promising but challenging technology.

References

[1] S. Rangan, T. S. Rappaport, and E. Erkip, "Millimeter-wave cellular wireless networks: Potentials and challenges," *Proceedings of the IEEE*, vol. 102, no. 3, pp. 366–385, 2014.

[2] A. Maltsev, A. Pudeyev, I. Karls, I. Bolotin, G. Morozov, R. Weiler, M. Peter, and W. Keusgen, "Quasi-deterministic approach to mmwave channel modeling in a non-stationary environment," in *Globecom Workshops (GC Wkshps), 2014*. IEEE, 2014, pp. 966–971.

[3] I. Filippini, V. Sciancalepore, F. Devoti, and A. Capone, "Fast cell discovery in mm-wave 5g networks with context information," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[4] D. Aziz, J. Gebert, A. Ambrosy, H. Bakker, and H. Halbauer, "Architecture approaches for 5g millimetre wave access assisted by 5g low-band using multi-connectivity," in *Globecom Workshops (GC Wkshps), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[5] C. Dehos, J. L. González, A. De Domenico, D. Ktenas, and L. Dussopt, "Millimeter-wave access and backhauling: the solution to the exponential data traffic increase in 5g mobile communications systems?" *IEEE Communications Magazine*, vol. 52, no. 9, pp. 88–95, 2014.

[6] A. Tolli, M. Codreanu, and M. Juntti, "Cooperative mimo-ofdm cellular system with soft handover between distributed base station antennas," *IEEE Transactions on Wireless Communications*, vol. 7, no. 4, pp. 1428–1440, 2008.

[7] H. Wang, C. Rosa, and K. Pedersen, "Performance analysis of downlink inter-band carrier aggregation in lte-advanced," in *Vehicular Technology Conference (VTC Fall), 2011 IEEE*. IEEE, 2011, pp. 1–5.

[8] M. Giordani, M. Mezzavilla, S. Rangan, and M. Zorzi, "Multi-connectivity in 5g mmwave cellular networks," in *Ad Hoc Networking Workshop (Med-Hoc-Net), 2016 Mediterranean*. IEEE, 2016, pp. 1–7.

[9] M. Polese, M. Giordani, M. Mezzavilla, S. Rangan, and M. Zorzi, "Improved handover through dual connectivity in 5g mmwave mobile networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 9, pp. 2069–2084, 2017.

[10] A. Ghosh, T. A. Thomas, M. C. Cudak, R. Ratasuk, P. Moorut, F. W. Vook, T. S. Rappaport, G. R. MacCartney, S. Sun, and S. Nie, "Millimeter-wave enhanced local area systems: A high-data-rate approach for future wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1152–1163, 2014.

[11] A. Maltsev, "Channel Modeling and Characterization - MiWEBA, Deliverable 5.1 EU Contract No. FP7-ICT-608637," https://www.miweba.eu, Tech. Rep., 2014.

[12] S. Sur, V. Venkateswaran, X. Zhang, and P. Ramanathan, "60 ghz indoor networking through flexible beams: A link-level profiling," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 71–84.

[13] K. Sundaresan, R. Sivakumar, M. A. Ingram, and T.-Y. Chang, "Medium access control in ad hoc networks with mimo links: optimization considerations and algorithms," *IEEE Transactions on Mobile Computing*, vol. 3, no. 4, pp. 350–365, 2004.

[14] A. Spyropoulos and C. S. Raghavendra, "Energy efficient communications in ad hoc networks using directional antennas," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 2002, pp. 220–228.

[15] Z. Huang, C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Topology control for ad hoc networks with directional antennas," in *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*. IEEE, 2002, pp. 16–21.

[16] A. Capone, I. Filippini, and F. Martignon, "Joint routing and scheduling optimization in wireless mesh networks with directional antennas," in *Communications, 2008. ICC'08. IEEE International Conference on*. IEEE, 2008, pp. 2951–2957.

[17] M. A. Guvensan and A. G. Yavuz, "On coverage issues in directional sensor networks: A survey," *Ad Hoc Networks*, vol. 9, no. 7, pp. 1238–1255, 2011.

[18] L. X. Cai, L. Cai, X. Shen, and J. W. Mark, "Rex: A randomized exclusive region based scheduling scheme for mmwave wpans with directional antenna," *IEEE Transactions on Wireless Communications*, vol. 9, no. 1, 2010.

[19] L. X. Cai, L. Cai, X. S. Shen, and J. W. Mark, "Resource management and qos provisioning for iptv over mmwave-based wpans with directional antenna," *Mobile Networks and Applications*, vol. 14, no. 2, pp. 210–219, 2009.

[20] L. X. Cai, H. Hwang, X. Shen, J. W. Mark, and L. Cai, "Optimizing geographic routing for millimeter-wave wireless networks with directional antenna," in *Broadband Communications, Networks, and Systems, 2009. BROADNETS 2009. International Conference on*. IEEE, 2009, pp. 1–8.

[21] Y. Niu, Y. Li, D. Jin, L. Su, and D. Wu, "Blockage robust and efficient scheduling for directional mmwave wpans," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 2, pp. 728–742, 2015.

[22] S. Singh, F. Ziliotto, U. Madhow, E. Belding, and M. Rodwell, "Blockage and directivity in 60 ghz wireless personal area networks: From cross-layer model to multihop mac design," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 8, 2009.

[23] M. Zhang, M. Mezzavilla, J. Zhu, S. Rangan, and S. Panwar, "Tcp dynamics over mmwave links," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, July 2017, pp. 1–6.

[24] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 619–632, 2012.

[25] J. Zhao and S. C. Sen-ching, "Optimal visual sensor planning," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 165–168.

Rethinking Service Chain Embedding for Cellular Network Slicing

Chrysa Papagianni
Institute for Systems Research
University of Maryland
College Park, Maryland, USA
chrisap@isr.umd.edu

Panagiotis Papadimitriou
Department of Applied Informatics
University of Macedonia, Greece
papadimitriou@uom.edu.gr

John S. Baras
Department of Electrical
and Computer Engineering
University of Maryland
College Park, Maryland, USA
baras@isr.umd.edu

***Abstract*—5G is set out to address the business contexts of 2020 and beyond, by enabling new network and service capabilities. The industry consensus is that 5G will facilitate ubiquitous connectivity, seamlessly integrating wireless technologies and complementary communication networks, while operators will be capable of providing networks on a need-for-service basis. Furthermore, there is a need for operators to exploit new revenue sources and break the traditional business model of a single network infrastructure ownership, by supporting multi-tenancy. Network slicing can provide a solution towards this end; it is considered a key for meeting 5G's diverse requirements, including future-proof scalability and flexibility.**

Provisioning and management of network slices in the transition from Long Term Evolution (LTE) to the emerging 5G systems poses the need for the mapping of service chains that express traffic and processing requirements of LTE slices. In this respect, we follow a different approach to the service chain mapping problem, promoting virtualized network function (NF) sharing among multiple service chains that are associated with a certain network slice. Using mixed-integer linear programming formulations, we show that our approach leads to reduced NF state and management overhead, compared to the common resource allocation practice in virtualized Radio Access Networks.

I. Introduction

Next-generation cellular networks will cater to a wide range of new business opportunities, such as network slice provisioning on a lease basis, in order to support multi-tenancy and meet diverse application requirements. Network Function Virtualization (NFV) and Software-Defined Networking (SDN) have been seen as key enablers towards 5G network slicing, as they allow the creation of customisable network elements which can be subsequently chained together programmatically. These network elements and functions can be easily configured and reused in each network slice to meet certain performance requirements, enabling new business opportunities by facilitating flexible and agile support for multi-service and multi-tenancy.

While the vision is very compelling from an infrastructure, operation and business perspective, the deployment of network slices poses various challenges, inherent to the enabling technologies, specific to the shared physical medium or associated to the application context. Focusing on Software Defined Mobile Networks (SDMN), different tenants issue requests to a mobile network provider for leasing network slices, where each slice as a logical end-to-end construct is self-contained, using network function chains for delivering services to a given group of devices.

ISBN 978-3-903176-08-9

Long Term Evolution (LTE) network slicing [1] commonly encompasses the following: (i) virtualizing the mobile core, deploying mobile core elements as virtualized network functions (vNF), and sharing the corresponding physical resources among tenants; (ii) sharing the base station (also termed as eNodeB) resources, where different scenarios can be supported for sharing physical resource blocks in the frequency/time/space domain at Layer 2; and (iii) sharing spectrum resources between different operators.

Considering the deployment of LTE elements as vNFs over virtualized infrastructures, authors in [2] introduce LTE as a Service framework, where both the mobile core services and eNodeB are deployed in a virtualized environment, using Openstack and Linux Containers. Authors in [1] describe the deployment of LTE Components as vNFs with OpenAirInterface (OAI) and the JUJU Framework, including mobile core network elements and 3GPP compliant eNodeBs, decomposed to the baseband unit (BBU) and remote radio head (RRH). Following the principles of the aforementioned approaches, we consider network slicing from the mobile core (termed as Evolved Packet Core - EPC) to the Radio Access Network (RAN), where virtualized eNodeBs are deployed, without, however, looking into aspects of slicing and apportioning the radio spectrum. Baseband processing functions are deployed on the virtualized eNodeBs, which are hosted on general-purpose hardware, supporting the dedicated RRHs implemented using software-defined radio (SDR) technology.

To ensure that network slices can deliver the desired benefits for each tenant, mobile network operators need to employ advanced techniques, which will optimize resource allocation for slice provisioning and also facilitate closed-loop performance maintenance. To this end, new algorithms and solutions need to be devised for allocating network and computing resources among different slices with the objective of meeting the performance and other functional requirements of applications/services, while, at the same time, maximizing the overall utility for the provider. In this respect, we consider a LTE slice composed of a group of service chains (SFC),

e.g., each one handling a set of traffic classes, such as voice, media streaming etc. Hence, we tackle this resource allocation problem at the granularity of a service chain, and, thereby, seek to optimize the assignment of service chains onto the virtualized RAN infrastructure. This essentially consists in the placement of virtulized LTE/EPC[1] elements (which are assumed to be implemented as vNFs) and the selection of the corresponding paths between these vNFs.

In most existing approaches (*e.g.,* [3], [4]), separate vNFs are allocated for each service chain, which means that each vNF instance is associated with a single chain. This approach yields: (i) increased overheads associated with vNF provisioning and management, (ii) potentially larger amount of NF state, if the state required by a LTE element is replicated among all the vNF instances in the slice, (iii) inefficient resource utilisation, since certain vNFs may have the required capacity to serve additional service chains, due to the statistical multiplexing of traffic, and (iv) fragmentation of resources, due to the larger number of vNF instances. To alleviate these inefficiencies, we promote the sharing of vNFs among the service chains of a LTE slice, aiming at lower provisioning and management costs as well as NF state reduction. To this end, we present mixed-integer linear programming (MILP) formulations for: (i) LTE service chain mapping with vNF sharing, and (ii) a baseline LTE service mapping that corresponds to the common resource allocation practice in virtualized RANs (*i.e.,* each service chain has its own dedicated vNFs).

The remainder of the paper is organized as follows. Section II provides an overview of the LTE network infrastructure. Section III describes the service chain mapping problem. In Section IV, we present our MILP formulations. In Section V, we compare the efficiency of our proposed method against the baseline using simulations. Section VI provides an overview of related work. Finally, in Section VII, we highlight our conclusions and discuss directions for future work.

II. Background

The term LTE encompasses the evolution of the UMTS radio access to the Evolved-UTRAN (E-UTRAN). This is accompanied by the evolution in the GPRS Core Network, under the name System Architecture Evolution (SAE), which includes the EPC network. LTE and SAE together constitute the Evolved Packet System.

E-UTRAN. The E-UTRAN consists of a network of base stations (termed as eNodeBs – eNBs) that provide radio access to the User Equipment (UE). eNBs provide user and control plane protocol terminations toward the UEs. They communicate with each other by means of the X2 interface. The eNBs are also connected via the S1 interface to the EPC. RANs are usually provisioned for peak loads, leading to inefficient resource utilisation, *i.e.,* up to 80% of CAPEX and 60% of OPEX in mobile networks is spent on RANs [5].

Centralised Radio Access Network (C-RAN) architecture splits the eNB to: (i) the BBU responsible for L1 digital processing of the baseband signal (*i.e.,* radio function) along with performing upper layer functions and interfacing with the backhaul; and (ii) the RRH performing functions such as amplification of RF signals, filtering, and AD/DA conversion. Following the C-RAN approach RRHs, installed close to the antenna, are connected to a centralized BBU pool at macro cell sites or central office locations, using the fronthaul transport network. Different protocols have been standardized for the fronthaul such as the common public radio interface (CPRI) [6]. Virtual RAN extends this flexibility further, through the virtualization of the execution environment [7], where radio functions is a network service running in a virtualized environment (Cloud RAN), potentially delivered as a cloud service (RAN as a service – RANaaS). Advances in the direction of leveraging NFV principles for C-RAN (also known as NFV C-RAN,vRAN) are currently emerging via proof of concept implementations [8].

EPC. The EPC contains user and control plane elements for routing, session establishment, mobility management, and billing. The user plane mainly consists of the Serving Gateway (S-GW) and the Packet Data Network Gateway (P-GW), used for UE traffic forwarding and tunnelling. More specifically, S-GW serves as a mobility anchor, whereas the P-GW routes UE traffic to external Packet Data Networks (PDNs). Mobility Management Entity (MME) is the main control plane element, responsible for UE authentication and authorization, session establishment and mobility management. The QoS level for each transmission path (termed as EPS bearer) between the UE and the P-GW is determined by the P-GW. When a UE is attached to the LTE, a default bearer is established supporting best-effort QoS. The EPS bearer consists of the radio data bearer (*i.e.* between UE and eNB), the S1 data bearer (*i.e.* between eNB and S-GW) and the S5 data bearer (*i.e.* between the SG-W and the P-GW). The GPRS tunneling protocol (GTP) is used for setting up the user plane datapaths between the eNB, S-GW and P-GW. In many cases, application-specific traffic (*e.g.* voice, video) is enforced to traverse a set of NFs, used by operators to differentiate their services [9]. Such NFs are traditionally deployed as specialized network devices, known as middleboxes. The SGi interface signifies the demarcation point between the EPC (P-GW) and the PDN. SGi-LAN refers to the NFs (*e.g.,* NATs, firewalls, caches) deployed by mobile operators on this reference point.

III. Problem Description

In order to create an LTE network slice, we should dynamically allocate, install, program and configure all the LTE network-specific elements. This requires the deployment of virtualized data and control plane functional entities (*i.e.,* BBU in RAN and MME, S/P-GW at the EPC) at the mobile operator's NFV Infrastructure (NFVI), which may span multiple NFVI Points-of-Presence (PoPs), *i.e.,* datacenters (DCs). NFVI-PoPs extend to the operator's WAN infrastructure, such as local or regional PoPs for small or larger-scale NFVI deployments. LTE network slicing further raises the need

[1]We refer to LTE/EPC as LTE in the rest of the paper.

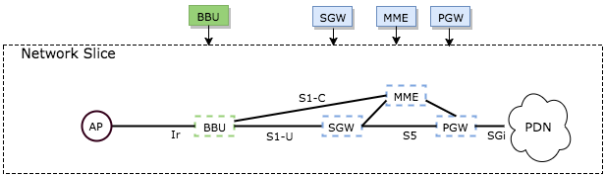


Fig. 1: LTE service chain.

for service chaining (*i.e.,* steering the traffic through a sequence of vNFs that implement the LTE user and control plane elements). Service chaining can be configured using flow tagging or tunneling to overcome the implications of "mangling" middleboxes, as exemplified in recent work [10], [11]. Fig. 1 illustrates such a service chain[2], whose end-points express different levels of abstraction at the mobile fronthaul (*e.g., Aggregation Point* for RRHs using appropriate equipment such as CPRI2Ethernet Gateway and CPRI mux). Based on this description, for a single LTE network slice, we need to efficiently place a set of LTE service chains, defined by the corresponding end-points (*e.g.,* one service chain per RRH aggregation point or RRH/cell).

The optimization problem at hand is the minimization of the resource provisioning cost for the LTE network operator, while allocating CPU and bandwidth for the LTE service chains. The problem is similar to service mapping (*e.g.,* [12]), since LTE service chains can be seen as bi-directional graphs that need to be embedded onto a substrate network [13]. This approach has been followed by recent work on NF placement in a virtualized EPC [4], [14]. However, in this way, the set of vNF forwarding graphs are mapped independently, leading to a potentially large number of vNFs, which in turn yields a substantial management cost for the LTE operator, especially during dynamic re-provisioning. Another downside of this approach is that the state required for each LTE element has to be replicated across a large number of NF instances, which essentially increases the total amount of state that has to be maintained.

In contrast to this common practice and inline with [15], we promote NF sharing across LTE service chains in order to reduce the number of NF instances and, consequently, the provisioning and management cost incurred by network slicing. In particular, we consider that flows from different cells (RRHs or Aggregation Points) can share and reuse NFs. For example, Fig. 2 illustrates two LTE service chains that share common vNFs (P-GW and S-GW). In this respect, we decompose the problem of *resource allocation for LTE network slicing* into the following steps:

- **Slice dimensioning**, which generates the number of NF instances (for each LTE data or control plane element) required to handle the expected traffic load. For example, a typical LTE system at a national level is composed of 10s of PGW, 100s of SGWs, and 1000s of eNBs [16]. The load is defined by the inbound traffic rate and the resource profile for each virtualized EPC functions (*i.e.* CPU cycles / packet).
- **NF placement**, which computes the optimized assignment of the generated NF instances onto the servers of the operator's NFVI.
- **Binding**, which associates the assigned NF instances with the LTE service chains, according to their computational and bandwidth requirements.
- **Path Selection**, which refers to the selection of the data paths through the LTE vNFs placed and bound to the service chains.

Following this approach, we present a MILP formulation for near-optimal LTE service chain mappings, by sharing vNFs among multiple LTE service chains. More specifically, vNF sharing is applicable, *e.g.,* for a set of LTE service chains on the same Tracking Area (TA), which is the logical grouping of neighbour eNBs in LTE networks, or the Tracking Area List (TAL), which is a group of Tracking Areas. TAs manage and locate UEs in a LTE network, when the UE is in *CONNECTED* state. However, in *IDLE* state the UE location is only known at TAL level. Therefore, at any point in time, the corresponding number of UEs in the TAL can provide an estimate of the expected load, which is required for slice dimensioning.

In Section IV, we present a baseline MILP formulation which corresponds to the common practice for LTE service chain mapping, *i.e.,* each LTE service chain is associated with its own individual vNFs. Section V provides a detailed comparison between the two methods and discusses the gains achieved by the MILP that promotes NF sharing.

IV. Problem Formulations for LTE Slicing

In this section, we discuss (i) the MILP formulation that shares NFs among service chains, and (ii) the baseline MILP formulation that allocates separate NFs per chain.

A. Service Mapping with NF Sharing

1) Network and Request Model: In the following, we introduce the network and request model for the MILP formulation with NF sharing.

Network Model. The operator has a number of $|A|$ available NFVI-PoPs interconnected via the provider's network. Each site's infrastructure is represented as a directed weighted graph $G_a = (N_a, E_a)$, where N_a represents the set of all nodes (*i.e.,* routers/switches, and servers) that belong to the operator's NFVI a and E_a the corresponding substrate links. Inter-DC links are denoted as $\{E_{aa'} = (u_a, v_{a'}) | \forall u_a \in N_a, v_{a'} \in N_{a'} \forall a, a' \in A, a \neq a'\}$. We consider a network-wide view of the operator's network; the overall substrate topology is denoted as $G_{S'} = (N_{S'}, E_{S'})$, where $N_{S'} = \cup_{a=1}^{A} N_a$ and $E_{S'} = \Big(\underset{\forall a \in A}{\cup} E_a \Big) \bigcup \Big(\underset{\substack{\forall a,a' \in A, \\ a \neq a'}}{\cup} E_{aa'} \Big)$.

We consider a set of I RRHs that belong to the same TAL. The length of the fronthaul link between the RRH and the BBU vNF can not exceed a given value; this guarantees the

[2]For the sake of readability, traffic flow direction in the uplink and the downlink is not depicted.

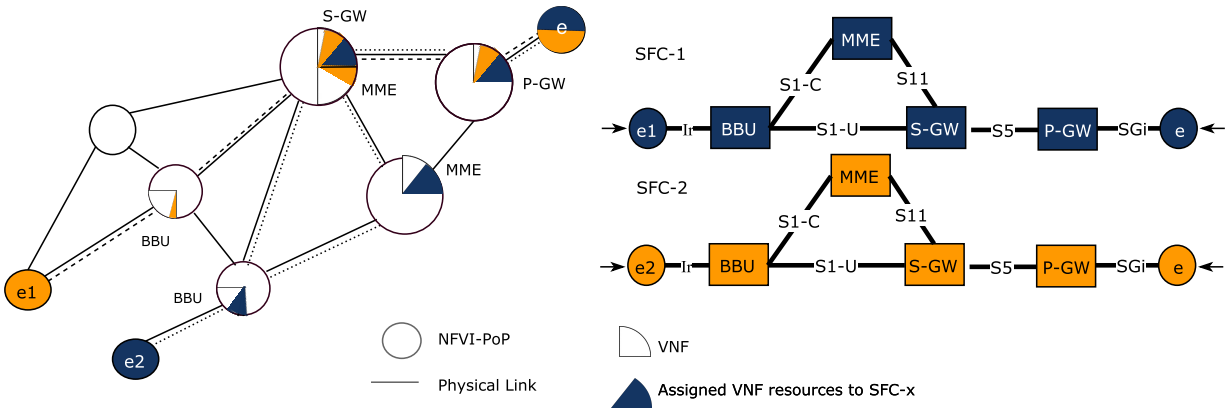


Fig. 2: LTE SFC(s) placed on physical infrastructure.

signal synchronization between RRH and BBU [17]. In this case, this is set to 20km [7]. We consider that there is at least one existing link from an RRH to an NFVI-PoP that meets this requirement. We augment the substrate graph node set with I pseudo nodes $N_S = N'_S \cup I$, (zero capacity). Network links between RRHs and PoPs are added to the link set, thus $E_S = E_{S'} \cup \{(u,i)|u \in N_{S'}, i \in I\} \cup \{(i,u)|u \in N_{S'}, i \in I\}$ forming the directed substrate graph $G_S = (N_S, E_S)$. Node sets of specific type are denoted as N_S^x (*i.e.,* routers, switches, RRHs, Internet Exchange Points (IXPs) and servers). Thus, the overall set of physical servers for the substrate is $N_S^{ser} = \cup_{a=1}^{A} N_a^{ser}$. Every node $n \in N_S^x$ and link $(u,v) \in E_S$ is associated with its residual capacity, denoted by r_n and r_{uv}, respectively. The residual capacity for substrate routers, switches and RRHs is set to zero.

Virtual Network Functions. The set N_V represents possible vNFs (*e.g.,* BBU, MME, P-GW, S-GW) that can be deployed at the NFVI-PoPs. Each NF instance is associated with a given amount of computing resources $d^i, i \in N_V$, used by the service chains assigned to that instance. Thus, inline with [15] we have NF instances of the same type (*e.g.,* set N_V^{MME} of MME instances) with different sizes. Each vNF $i \in N_V$ can be instantiated at a substrate node of type N_S^{ser} at most U_i times (*e.g.,* depending on the number of purchased licenses). We extend the set N_V with two additional pseudo vNFs, RRH and IXP, assuming they can be instantiated only at N_S^{RRH} and N_S^{router}, respectively, utilizing minimal computing resources.

Service Chain Model. We use a directed weighted graph $G_f = (N_f, E_f), f \in F$ to express each service chain request, where F represents the set of SFCs. The set of vertices N_f includes two sets: (i) N_f^V: the set of vNFs that belong to either the RAN or the EPC, as well as any NFs (*e.g.,* NAT, firewall) that the traffic has to traverse; and (ii) N_f^S: the set of service chain end-points (RRH and IXP, in this case). Each vertex $k \in N_f$ is associated with a computing demand $g^{f,k}$, which we estimate based on the inbound traffic rate and the resource profile of the LTE element (*i.e.,* CPU cycles / packet), apart from the endpoints (N_f^S) where the computing resources are minimal. The edges are denoted by $(k,m) \in E_f$ while their bandwidth demands are expressed by $g^{f,km}$ for SFC $f \in F$. We further introduce $l_u^{f,k}$ which represents the distance between the preferred location of a function $k \in N_f^S, f \in F$ and the location of the server where this will be hosted, with $u \in N_S$.

2) Problem Formulation: In the MILP formulation, the binary variable $x_u^{i,j}$ expresses the placement of instance j of vNF $i \in N_V$ on the substrate node $u \in N_S$. Furthermore, the binary variable $z_u^{f,k}$ indicates the assignment of vNF $k \in N_f$ required by service chain $f \in F$ to the substrate node $u \in N_S$. The real variable $f_{uv}^{f,km}$ expresses the amount of bandwidth assigned to link $(u,v) \in E_S$ for graph edge (k,m) in the vNF forwarding graph of service chain $f \in F$.

Objective:

$$\text{Min.} \sum_{i \in N_V} \sum_{j \in U_i} \sum_{u \in N_S} d^i x_u^{i,j} + \sum_{f \in F} \sum_{(k,m) \in E_f} \sum_{\substack{(u,v) \in E_S \\ (u \neq v)}} f_{uv}^{f,km} \quad (1)$$

Capacity related Constraints:

$$\sum_{\forall i \in N_V} \sum_{j \in U_i} d^i x_u^{i,j} \leq r_u \quad \forall u \in N_S \quad (2)$$

$$\sum_{f \in F} \sum_{(k,m) \in E_f} f_{uv}^{f,km} \leq r_{uv} \quad \forall (u,v) \in E_S \quad (3)$$

Placement and Assignment related Constraints:

$$\sum_{\forall i \in N_V^{x'}} \sum_{j \in U_i} x_u^{i,j} = 0 \quad \forall u \in N_S^x, x' \neq x \quad (4)$$

$$\sum_{\forall u \in N_S^x} x_u^{i,j} \leq 1 \quad \forall i \in N_V^x, j \in U_i \quad (5)$$

$$\sum_{\substack{\forall f \in F \\ k \in N_f : k = i}} g^{f,k} z_u^{f,k} \leq \sum_{i' \in N_V : i' = i} \sum_{j \in U_i'} d^{i'} x_u^{i',j} \quad \forall u \in N_S, i \in N_V \quad (6)$$

$$z_u^{f,k} \leq \sum_{\substack{j \in U_i \\ i \in N_V : i = k}} x_u^{i,j} \quad \forall k \in N_f, f \in F, u \in N_S \quad (7)$$

$$\sum_{u \in N_S} z_u^{f,k} = 1 \quad \forall k \in N_F, f \in F \quad (8)$$

$$l_u^{f,k} z_u^{f,k} = 0 \quad \forall k \in N_f^S \subset N_f, f \in F, \forall u \in N_S \quad (9)$$

Flow related Constraints:

$$\sum_{\substack{v\in N_S\\(u\neq v)}} (f_{uv}^{f,km} - f_{vu}^{f,km}) = g^{f,km}(z_u^{f,k} - z_u^{f,m})$$
$$m \neq k, \forall (m,k) \in E_f, f \in F, u \in N_S \quad (10)$$

Domain Constraints:

$$x_u^{i,j} \in \{0,1\} \quad \forall i \in N_V, j \in U_i, u \in N_S \quad (11)$$
$$z_u^{f,k} \in \{0,1\} \quad \forall k \in N_f, f \in F, u \in N_S \quad (12)$$
$$f_{uv}^{f,km} \geq 0 \quad \forall (u,v) \in E_S, (k,m) \in E_f, f \in F \quad (13)$$

The optimization objective of the MILP is expressed by the objective function (1). The first term of this function represents the CPU requirements, based on the vNF instances mapped to the infrastructure. The second term of the objective function expresses the accumulated bandwidth assigned to substrate links. Constraint (2) ensures that the sum of CPU required by the vNF instances mapped to substrate node u does not exceed the residucal processing power. Constraint (3) ensures that the allocated bandwidth does not exceed the residual capacity of links. Condition (4) enforces the placement of vNF (and pseudo vNF) instances on substrate nodes that meet the vNF's functional requirements. Constraint (5) ensures that each vNF instance is placed at a single substrate node. Constraint (6) ensures that the sum of processing demands of service chain elements does not exceed the amount of virtual resources provided by vNFs of type i mapped to substrate node u. Constraint (7) ensures that, if a vNF requested by an service chain is assigned to substrate node u, then at least one instance should be placed on u. Constraint (8) ensures that every required service chain (and its respective vNFs) is mapped to the infrastructure. Condition (9) enforces location constraints for the service chain endpoints. Constraint (10) enforces flow conservation, *i.e.,* the sum of all inbound and outbound traffic in switches, routers, and servers that do not host vNFs should be zero. More precisely, this condition ensures that for a given pair of assigned nodes k,m (*i.e.,* vNFs or end-points), there is a path in the network graph where the edge (k,m) has been mapped. Finally, conditions (11), (12) and (13) express the domain constraints for the three variables.

We note that the complexity of the proposed MILP can be reduced by: (i) relaxing the integer domain constraints, and (ii) using a rounding algorithm to extract feasible solutions from non-integer values. Rounding can be performed by employing existing deterministic and randomized techniques used in service mapping [18], [4], [12]. Due to space limitations, we leave this for future work.

B. Baseline Service Mapping

In the following, we discuss the MILP formulation for the baseline service mapping without NF sharing.

1) Request Model: As the *Network Model* is similar to the one described above, we hereby present only the *Request Model*.

Request Model. We use a directed graph $G_F = (N_F, E_F)$ to express a service chain request. The set of vertices N_F includes two sets: (i) N_F^V: the set of vNFs that belong to either the RAN or the EPC, and any other vNFs for additional processing; and (ii) N_F^S: the set of service chain end-points. Each vNF $i \in N_F^V$ can be instantiated at a substrate node of type N_S^{ser}, while RRH and IXP can be instantiated only at the corresponding N_S^{RRH} and N_S^{router}, respectively. Each vertex N_F^V in the graph is associated with a computing demand g^i. The edges are denoted by $(i,j) \in E_F$ while their bandwidth demands are expressed by g^{ij}. We also use l_u^i, as defined in the service chain model in Section IV-A.

2) Problem Formulation: In the following MILP formulation, we use the binary variable x_u^i to express the placement of vNF $i \in N_F$ of the service chain request G_F on the substrate node $u \in N_S$. The real variable f_{uv}^{ij} expresses the amount of bandwidth assigned to link $(u,v) \in E_S$ for the vNF graph edge (i,j).

Objective:

$$\text{Min.} \sum_{i\in N_F}\sum_{u\in N_S} x_u^i + \frac{1}{\sum_{(i,j)\in E_F} g^{ij}} \sum_{(i,j)\in E_F} \sum_{\substack{(u,v)\in E_S\\(u\neq v)}} f_{uv}^{ij} \quad (14)$$

Capacity related Constraints:

$$\sum_{\forall i\in N_F} g^i x_u^i \leq r_u \quad \forall u \in N_S \quad (15)$$
$$\sum_{\forall (i,j)\in E_F} f_{uv}^{ij} \leq r_{uv} \quad \forall (u,v) \in E_S \quad (16)$$

Placement related Constraints:

$$\sum_{\forall i\in N_F^{x'}} x_u^i = 0 \quad \forall u \in N_S^x, x' \neq x \quad (17)$$
$$\sum_{\forall u\in N_S^x} x_u^i = 1 \quad \forall i \in N_F^x \quad (18)$$
$$l_u^i x_u^i = 0 \quad \forall i \in N_F^S \subset N_F, \forall u \in N_S \quad (19)$$

Flow related Constraints:

$$\sum_{\substack{v\in N_S\\(u\neq v)}} (f_{uv}^{ij} - f_{vu}^{ij}) = g^{ij}(x_u^i - x_u^j) \quad i \neq j, \forall (i,j) \in E_F, u \in N_S \quad (20)$$

Domain Constraints:

$$x_u^i \in \{0,1\} \quad \forall i \in N_F, u \in N_S \quad (21)$$
$$f_{uv}^{ij} \geq 0 \quad \forall (u,v) \in E_S, (i,j) \in E_F \quad (22)$$

The optimization objective of the MILP is expressed by the objective function (14). The first term of this function represents the number of assigned servers. The second term of the objective function expresses the accumulated bandwidth assigned to substrate links divided by the total bandwidth demand. Constraint (15) ensures that the sum of processing demands of the vNFs mapped to substrate node u does not exceed the residual computing capacity. Constraint (16) ensures that the allocated bandwidth resources do not exceed the

residual link capacity. Condition (17) enforces the placement of vNFs (and pseudo vNFs) on substrate node types that meet the vNF's functional requirements. Constraint (18) ensures that a vNF is placed at most on a substrate node. Condition (19) enforces location constraints for the service chain endpoints. Constraint (20) enforces flow conservation. Finally, conditions (21) and (22) express the domain constraints for the variables. Similar to the previous MILP formulation, relaxation and rounding techniques can be employed to reduce the time complexity.

V. Evaluation

In this section, we evaluate the efficiency and discuss the feasibility of the proposed MILP model, denoted as *NF-Sharing*. The model is compared against the *Baseline* service mapping model without NF sharing. In the following we discuss the evaluation environment (Section V-A), the evaluation metrics (Section V-B) and the evaluation results (Section V-C).

A. Evaluation Environment

We have implemented an evaluation environment in Java including a service chain generator and a cellular network topology generator. We use CPLEX for our MILP models using the branch-and-cut method. Our tests are carried out on a server with one Intel Xeon four-core CPU at 3.5 GHz and 6 GB of allocated main memory.

Given the time complexity of the mixed-integer linear programs, we use a small-scale LTE scenario for the validation/evaluation of the proposed models, based on the real-world scenario presented in [19] that was created using real statistics from a region in Paris, while LTE SFCs are jointly mapped at the Tracking Area List level considering however a single TA per TAL.

NFV Infrastructure. Similar to [4], we have generated a PoP-level network topology with homogeneous NFVI PoPs. Each PoP is essentially a micro-DC with a two-level fat-tree network topology. Table I shows additional NFVI parameters. Regarding the vNF instances for *NF-Sharing*, we consider three distinct levels of LTE vNFs that can support up to 500, 750 and 1000 UEs respectively.

E-UTRAN. We rely on a multi-cell scenario for the RAN. Table II presents the E-UTRAN design parameters. Considering uniform circular cells with an overlapping factor γ of 1.2, the cell radius is $r = \gamma\sqrt{A_t/c\pi}$ (approximately 0.64 km for the above-mentioned settings). In our case, we consider varying user density (up to $rho = 385 UEs/km^2$), so that the number of active UEs per eNB ranges from 200 to 500. We also provide the number of RRHs at Tracking Area level and the Tracking Area size. The maximum length for the RRH-BBU link is limited to 20 km [8].

Traffic Classes. Similar to [4], traffic is classified into three types, *i.e.,* voice, media streaming, and background traffic, with their busy-hour parameters shown in Table III [19] [4]. $Pr\{O\}$ is the probability that a session of a specific application type is originated by a UE.

LTE vNF profiles. The CPU demand for each vNF can be derived based on the inbound traffic rate and the resource profile of the vNF (*i.e.,* CPU cycles per packet). Resource profiles are available for a wide range of NFs (*e.g.,* IPv4 forwarding [20], [21]), while existing profiling techniques (*e.g.,* [22]) can be applied to any flow processing workloads whose computational requirements are not known. We derive the CPU demands for each NF from resource profiles, similar to [12] [4]. We extract the resource profile of the MME using the study on the latency evaluation of a virtualized MME [23]. The BBU processing budget of a GPP platform was based on the study by Nikain on OAI implementation [8] that considers three functions as the main contributors to the BBU processing budget namely; iFFT/FFT, (de)modulation, and (de)coding. The proposed model computes the total BBU uplink and downlink processing time for different physical resource blocks, modulation and coding scheme (MCS) and virtualization environment. We use the particular model considering an Intel SandyBridge architecture with a CPU frequency of 3.2GHz, a channel bandwidth of 20 MHz assuming 64 quadrature amplitude modulation (QAM) in the downlink and 16 QAM in the uplink and Linux Containers platform. This leads to a total processing time of 723.5 us per subframe in the downlink and 1062.4 us per subframe in the uplink.

Service Chains. We generate vNF-forwarding graphs per cell according to Fig. 1 class based on service chain templates. In particular, each service chain contains the main LTE elements (*i.e.,* BBU, S/P-GW, MME) using the aforementioned NF profiles.

Signalling Load and Traffic. We quantify the processing load and the uplink/downlink traffic generated by LTE/EPC data management procedures, using the aforementioned traffic profile based on the analysis provided in [19] and 3GPP LTE/EPC signalling messages and their sizes provided in [24]. In this respect, applications are modelled as ON-OFF state machines, while we assume that each UE is registered in the LTE/EPC network (EMM-registered) and alternates between Connected (*ECM-Connected*) and Idle (*ECM-Idle*) states. In other words, only *Service Request/Release* procedures are taken into account. The RRC inactivity timer defines the inactivity period required for the UE to switch to IDLE state. This timer is adjusted to 40 sec, which is a widely used setting in cellular networks [19].

B. Evaluation Metrics

We use the following metrics for the evaluation of the two service chain mapping methods:

- **vNF Instances** expresses the number of vNF instances that need to be instantiated (which is strongly correlated with the amount of vNF state) in order to support the embedded SFCs.
- **Hop Count** of the vNF forwarding graph edge expresses the length of the physical path where the edge is mapped.
- **Load Balancing Level (LBL)** is defined as the maximum over the average load. We report the (moving average)

TABLE I: NFVI Parameters

NFVI PoPs	2
Servers per DC	20 in 2 racks
Server Capacity	$24 \cdot 3.2$ GHz
ToR-to-Server link capacity	8 Gbps
Inter-rack link capacity	32 Gbps
Inter-DC link capacity	100 Gbps

TABLE II: User Modeling Parameters

Area Size (A_t)	180 km^2
Total Number of eNBs in the area (C)	200
Active UEs per eNB	200 ...500
Tracking Area Size	9 km^2
Total Number of eNBs in TA	10

TABLE III: Session Parameters

Application Type	Arrival Rate (1/hour)	Duration (seconds)	Nominal Rate (Kbps)	Pr(0)
Voice	0.67	180	23.85	0.5
Streaming	5	180	2500	1
Background traffic	40	10	1500	0.8

LBL for DCs based on server load. Lower values of LBL represent better load balancing, while $LBL = 1$ designates optimal load balancing.

- **Request Acceptance Rate** is the ratio of successfully embedded requests over the total number of requests.
- **Revenue per Request** is the amount of CPU and bandwidth units specified in the request. In this case we present the aggregated revenue of the successfully embedded SFC requests.

C. Evaluation Results

Fig. 3 illustrates the number of LTE vNF instances used to serve the incoming requests, Fig. 4 depicts the CDF of the hop count of vNF graph edges mapped to physical paths (when all the vNFs of a service chain are collocated we consider the hop count to be 0), whereas Fig. 5 plots the load balancing level across DCs. The NF-Sharing approach reduces significantly (approximately by $\tilde{4}7\%$) the number of vNFs assigned at the NFVI, reducing as a result the corresponding management overhead and provisioning costs associated with vNF instances. According to Fig. 4, the baseline approach employs distinct NF instances per service chain and collocates the vNFs of a service chain in the same host more often than the NF-sharing approach, as means to decrease the cost of the objective function. The behavior of the NF-Sharing approach is consistent with its formulation, attempting at every opportunity to minimize the number of vNF instances used by the incoming batch of service chain requests. However, consolidation leads to a slightly larger number of service chains assigned to vNFs that are placed on different servers; hence, the difference among the hop count CDFs. Therefore, embedding with NF-sharing increases the number of hops onto which vNF graph edges are mapped, although a larger instance of the problem would provide more insight on the particular aspect. At the same time, the NF-sharing approach yields better load balancing, comparing the corresponding load balancing levels for DC1 and DC2 with the baseline.

Fig 6 and 7 illustrate the request acceptance rate of the two approaches, and the corresponding revenue from embedding the service chains, respectively. The baseline leads to an increased acceptance rate and corresponding revenue, due to its intrinsic flexibility, placing independently vNFs per chain. When the DC utilization level increases significantly, the NF-Sharing approach cannot map the corresponding set of service chains per TAL, as opposed to the baseline that embeds chains with finer granularity (approximately 10% higher than NF-Sharing). However, flexibility comes at the cost of a larger number of vNF instances. NF-Sharing results in a trade-off by reducing the number of vNFs assigned at the NFVI, with a proportionally quite smaller reduction at the acceptance rate and revenue. Certainly, a high request acceptance rate is important for the infrastructure provider, since he can increase his revenue by meeting the requirements of multiple Mobile Virtual Network Operators (MVNO) that lease network slices. However, in the process of providing LTE as a Service, operational costs (*e.g.,* slice provisioning/configuration, as the NF state is significantly less with the NF-Sharing approach) need also to be taken into consideration.

Our goal is to decompose the LTE network elements into vNF instances that are easily instantiated based on capacity requirements, but without over-fragmentation that increases the overheads associated with provisioning and NF state management; that is exactly what the NF-sharing approach achieves. Furthermore, optimizing NF placement is particularly important in a dynamic environment where resources become fragmented over time, and it might not be possible for all VNFs in a service chain to be placed in proximity. Based on our results, we believe that the enforced policy on NF placement can potentially change over time in order to reap the benefits of both solutions. More precisely, the NF-Sharing approach is deemed more appropriate for low and medium utilization levels in order to reduce vNF state, while the baseling can be employed under high utilization to exploit its flexible NF placement that eventually leads to higher acceptance rate and revenue.

VI. Related Work

In this section, we discuss related work on EPC and RAN virtualization.

EPC. Research has been conducted on the instantiation of LTE mobile core gateways (S-GW and/or PGW) as vNFs [25], [26], [27]. Alternative approaches in the same direction take into consideration data-plane delay constraints [28], [29]. However, the aforementioned methods optimize the placement only of data-plane functions for various objectives (*e.g.,* minimizing the EPC resource provisioning cost, load balancing). Recently, control-plane EPC NF placement (*e.g.,* MME, PCRF, HSS) along S/P-GWs has been also considered towards a 3GPP-compliant elastic cellular core [3], [14]. In addition, [30], [4] propose MILP formulations for the joint embedding of core

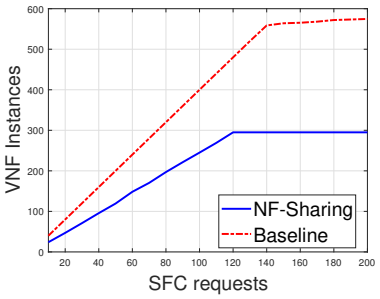


Fig. 3: Number of vNF Instances.

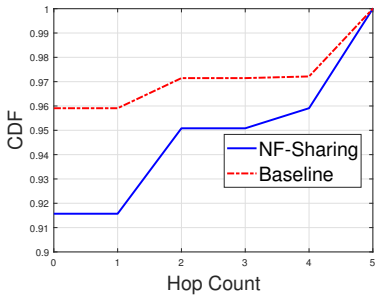


Fig. 4: CDF of hop count

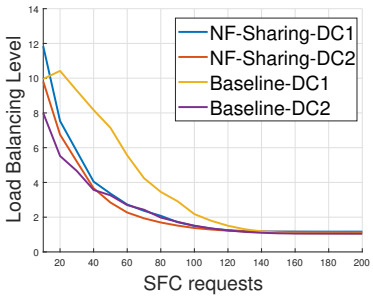


Fig. 5: DC load balancing level.

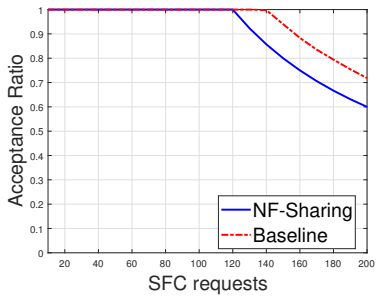


Fig. 6: Request Acceptance Rate.

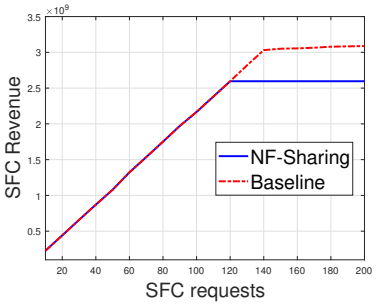


Fig. 7: Aggregated Revenue.

network service chains, taking into account latency budgets between communicating EPC elements, according to 3GPP.

KLEIN [3] presents a orchestration platform for EPC virtualization aiming at load balancing across the operator's datacenters. In terms of NF placement on the virtualized EPC, KLEIN decomposes the placement optimization into three steps (*i.e.,* region, datacenter, and server selection) to cope with the problem complexity at large scale.

RAN. There has been comprehensive research over the past years on minimizing energy consumption in RAN. The problem has been formulated as a joint optimization of RRH selection and power-minimization beamforming [31], [32] or joint optimization of RRH selection and precoding design [33], [34]. Shifting the focus towards the fronthaul and efficient resource usage of the BBU pool, BBU placement has been jointly optimized with the fronthaul transport network [35], [36], [37]. However, these studies are not focused on the placement of virtualized RAN elements.

Following a technology-agnostic approach the problem of BBU placement and RRH assignment in the RAN has been recently investigated. Authors in [38] address the problem in the context of a virtualized RAN, where functions from an eNB (*e.g.,* BBU) are implemented in a shared infrastructure located at either a DC or distributed in network nodes. Specifically this work attempts to minimize (i) the deployment cost of a BBU server, (ii) the cost of setting up the fronthaul links required between the BBUs and RRHs, and (iii) the deviation between the desired and actual latency in the fronthaul links, subject to constraints related to the resource capacities of the physical resources and a corresponding budget for the maximum number of BBU servers. The ILP formulation can be reduced to the maximal covering location problem that is known to be NP-hard. The authors propose a cost-aware greedy algorithm, reaching potentially a suboptimal placement and assignment solution, through a ranking and selection procedure. Authors in [17] strive to minimize the cost of deploying a BBU pool increased by the cost of the corresponding fronthaul links required, while respecting the resource capacities of the physical resources and ensuring that the length of the optical link between the BBU pool and RRHs for signal synchronization can not exceed a predefined maximum value. The problem is formulated as an ILP and solved using a local search heuristic.

In contrast to the aforementioned studies, we provide optimization methods for the joint placement of E-UTRAN and EPC elements onto virtualized infrastructures, as an enabler for 5G network slicing. Our approach is also different, as we enable NF sharing among service chains in order to reduce the number of NF instances and, consequently, the associated provisioning and management cost for cellular network operators.

VII. Conclusions

Towards the delivery of LTE as a service, we tackled the challenging problem of LTE service chain assignment onto the operator's NFV infrastructure, from a different perspective. In this respect, we proposed a MILP formulation for near-optimal LTE service chain mappings, by sharing vNFs among multiple service chains in a network slice, as means to reduce the provisioning and management cost (which is strongly correlated with the number of vNF instances), as well as the fragmentation of resources. To identify potential gains

stemming from vNF sharing, we compared our proposed MILP against a baseline MILP which assigns separate vNFs for each service chain.

Our evaluation results corroborate the smaller number of vNF instances allocated with the proposed MILP. This essentially leads to lower overheads with respect to vNF provisioning and management. Additional gains brought by NF sharing include the reduction in the path length and better load balancing in the operator's DCs. Our evaluation further indicates that these gains diminish at high utilization levels, at which the flexibility afforded by a larger number of vNF instances may be preferable by the operator, as it can lead to higher request acceptance rates, and thereby, larger generated revenue. Our evaluation can be used to drive the development of a hybrid LTE service chain mapping approach, at which NF sharing can be enabled depending on the DC load.

In future work, we plan to conduct an experimental evaluation of NF sharing in virtualized RANs in order to quantify the provisioning and management cost savings for the operator. We will further investigate whether NF sharing introduces any implications on resource isolation among the different service chains.

ACKNOWLEDGMENTS

This work is partially supported by the EU-BRA Horizon 2020 NECOS Project (Grant Agreement No. 777067).

REFERENCES

[1] K. Katsalis et al, "Network slices toward 5G communications: Slicing the LTE network," IEEE Communications Magazine, vol. 55, no. 8, 2017, pp.146-154.

[2] N. Nikaein et al., "Network Store: Exploring Slicing in Future 5G Networks," ACM MobiArch, Paris, France, Sep. 2015.

[3] Z. Qazi et al., "KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core," ACM SOSR '16, Santa Clara, CA, USA, March, 2016.

[4] D. Dietrich et al., "Network Function Placement on Virtualized Cellular Cores," IEEE COMSNETS, Bangalore, India, January 2017.

[5] China Mobile Research Institute,"C-RAN The Road Towards Green RAN," White Paper. Version 2.5. [Online]. Available: http://labs.chinamobile.com/cran/wpcontent/uploads/CRAN_white_paper_v2_5_EN.pdf

[6] FUJITSU, "The Benefits of Cloud-RAN Architecture in Mobile Network Expansion", [Online]. Available: http://www.fujitsu.com/downloads/TEL/fnc/whitepapers/CloudRANwp.pdf.1 [Accessed: Dec. 8, 2017].

[7] N. Nikaein. "Processing Radio Access Network Functions in the Cloud: Critical Issues and Modeling," ACM MCS '15, Paris, France, Sept. 2015.

[8] N. Nikaein, et al., "Demo: Closer to Cloud-RAN: RAN as a Service," ACM MobiCom '15, Paris, France, Sept. 2015.

[9] IETF Service Function Chaining Use Cases in Mobile Networks. [Online]. Available: https:tools.ietf.orghtmldraft-ietf-sfc-use-case-mobility-02 [Accessed: Dec. 8, 2017].

[10] S. Fayazbakhsh et al., Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags, ACM SIGCOMM HotSDN '13, Hong Kong, China, August 2013.

[11] Z. Qazi et al., "SIMPLE-fying middlebox policy enforcement using SDN," ACM SIGCOMM '13, Hong Kong, China, August 2013.

[12] D. Dietrich et al., "Multi-Provider Service Chain Embedding with Nestor," IEEE Transactions on Network and Service Management, vol. 14, no. 1, March 2017, pp. 91-105.

[13] S. Mehraghdam, M. Keller, and H.Karl, "Specifying and placing chains of virtual network functions," CloudNet, London, UK, Oct. 2014.

[14] A. Baumgartner, V.S. Reddy, and T. Bauschert, "Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization," IEEE NetSoft, London, UK, June 2015.

[15] M. C. Luizelli et al., "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," IFIP/IEEE IM, Ottawa, Canada, July 2015.

[16] SONATA D2.1 Use Cases and Requirements. [Online]. Available: http://www.sonata-nfv.eu/content/d21-use-cases-and-requirements [Accessed: Dec. 8, 2017].

[17] S. Xu and S. Wang, "Efficient Algorithm for Baseband Unit Pool Planning in Cloud Radio Access Networks," VTC 2016, Nanjing, China, May 2016.

[18] M. Chowdhury, M. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," IEEE/ACM Transactions on Networking, vol. 20, no. 1, Feb. 2012, pp. 206-219.

[19] W. Diego, I. Hamchaoui, and X. Lagrange, "The Cost of QoS in LTE/EPC Mobile Networks Evaluation of Processing Load," VTC2015, Boston, MA, USA, Sep. 2015.

[20] A. Abujoda, and P. Papadimitriou, "Profiling packet processing workloads on commodity servers", IFIP WWIC 2013, St. Petersburg, Russia, June 2013.

[21] M. Dobrescu, K. Argyarki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," USENIX NSDI, San Jose, CA, USA, March 2016.

[22] Q. Wu and T. Wolf, "Runtime Task Allocation in Multi-Core Packet Processing Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 10, Oct. 2012, pp. 1934–1943.

[23] J. Prados-Garzon et al., "Latency evaluation of a virtualized MME," IEEE Wireless Days, Toulouse, France, March 2016.

[24] M. R. Sama et al., "Enabling network programmability in LTE/EPC architecture using OpenFlow," IEEE WiOpt, Hammamet, Tunisia, May 2014.

[25] T. Taleb and A. Ksentini, "Gateway relocation avoidance-aware network function placement in carrier cloud," ACM MSWiM '13, Barcelona, Spain, Nov. 2013.

[26] M. Bagaa, T. Taleb, and A. Ksentini, "Service-Aware Network Function Placement for Efficient Traffic Handling in Carrier Cloud," IEEE WCNC, Istanbul, Turkey, Nov. 2014.

[27] F. Yousaf et al., "SoftEPC: Dynamic instantiation of mobile core network entities for efficient resource utilization," IEEE ICC, Budapest, Hungary, June 2013.

[28] A. Basta et al., "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem," ACM AllThingsCellular, Chicago ,IL, USA, August 2014.

[29] T. Taleb, M. Bagaa, and A. Ksentini, "User mobility-aware virtual network function placement for virtual 5G network infrastructure," IEEE ICC, London, UK, June 2015.

[30] A. Baumgartner, V.S. Reddy, and T. Bauschert, "Combined Virtual Mobile Core Network Function Placement and Topology Optimization with Latency Bounds," IEEE EWSDN, Bilbao, Spain, Sept. 2015.

[31] Y. Shi, J. Zhang, and K.B. Letaief, "Group sparse beamforming for green Cloud-RAN," IEEE/ACM Transactions on Wireless Communications, vol. 13, no. 5, May 2014, pp. 2809-2823.

[32] J. Tang, W.P. Tay, and T.Q. Quek, "Cross-layer resource allocation with elastic service scaling in cloud radio access network," IEEE/ACM Transactions on Wireless Communications, vol. 14, no. 9, Sept. 2015, pp. 5068-5081.

[33] V.N. Ha, L.B. Le, and Ngo. c-Dung Dao, "Cooperative transmission in cloud RAN considering fronthaul capacity and cloud processing constraints," IEEE WCNC, Istanbul, Turkey, Nov. 2014.

[34] V.N. Ha and L.B. Le, "Computation capacity constrained joint transmission design for CRANs," IEEE WCNC, Doha, Qatar, April 2016.

[35] F. Musumeci et al., "Optimal BBU placement for 5G C-RAN deployment over WDM aggregation networks," Journal of Lightwave Technology, vol. 34, no. 8, April 2016, pp. 1963-1970.

[36] A. Asensio et al., "Study of the Centralization Level of Optical Network-Supported Cloud RAN," IEEE ONDM, Cartagena, Spain, May 2016.

[37] K. Sundaresan et al., "Fluidnet: A flexible cloud-based radio access network for small cells," IEEE/ACM Transactions on Networking, vol. 24, no. 2, April 2016, pp. 915-928.

[38] R. Mijumbi et al, "Server placement and assignment in virtualized radio access networks," IEEE CNSM 2015, Barcelona, Spain, Nov. 2015.

D2D Multihop Energy-Efficient Routing and OFDMA Resource Allocation in 5G Networks

Safwan Alwan*, Ilhem Fajjari† and Nadjib Aitsaadi‡
*University Paris-Est, LiSSi EA 3956, UPEC, F94400, Vitry-sur-Seine, France
†Orange Labs, F92320, Chatillon, France
‡University Paris-Est, LIGM-CNRS UMR 8049, ESIEE Paris, F93160, Noisy-le-Grand, France
Emails: safwan.alwan@univ-paris-est.fr, ilhem.fajjari@orange.com, nadjib.aitsaadi@esiee.fr

***Abstract*—In face of the rapidly increasing cellular traffic, 5G will employ offloading techniques to relieve the cellular infrastructure. The idea is to carry the traffic locally by User-Equipment (UE) relays or by other coexisting radio access technologies such as WiFi, Bluetooth, etc. To this end, in this paper, we propose an offloading scheme using multi-hop LTE-D2D communications and assisted by the operator. LTE-D2D UEs cooperate to carry intra-cell unicast/multicast traffic from sources to destinations by exploiting i) sidelink interfaces and ii) multi-hop paths. To increase the lifetime of the offloading system and to reduce the impact of the relaying process on the battery-limited UEs, we propose our energy-aware approach, named `JRRA-EE`, to solve jointly the routing and the OFDMA resource block allocation. We formulate our problem as a 0-1 Integer Linear Programming (ILP) model which is elaborated to take into consideration the realistic LTE-D2D capabilities and constraints. To gauge the effectiveness of our proposal, we implement the whole 3GPP LTE-D2D protocol stack in the NS-3 network simulator to simulate our approach. Based on extensive simulations, the obtained performances of `JRRA-EE` are better compared to other one-sided optimal strategies, including an energy non-aware variant, in terms of i) the network lifetime, ii) the packet loss and iii) the service interruption rate.**

***Index Terms*—LTE-D2D, Routing, OFDMA resource block allocation, Energy-aware offloading, Optimization.**

I. Introduction

In recent years, the Device-to-Device (D2D) communication paradigm has received much attention of the academic and industrial communities. Relying on the physical proximity of user terminals, D2D offers low-energy cost and short-distance communications. Furthermore, D2D allows reusing the existing of classical cellular hardware as well as the same frequency resources leading to improve the overall network and spectral efficiency. Apart from these advantages, D2D communication also enables many new applications such as proximity-based safety and commercial services, cooperative content sharing and relaying.

In this paper, we tackle both the routing and OFDMA resource block allocation for an energy-efficient offloading mechanism within the LTE networks for multicast and/or unicast flow-oriented application. Note that, from a formulation point of view, a unicast flow is a special case of a multicast (i.e., one destination). Specifically, within a single LTE eNodeB cell, our system utilizes the sub-network of LTE-D2D-enabled User Equipments (UEs) to route flows that originates from and terminates in the same macro-cell. As a matter of fact, while the data plane offloading rests with the UEs themselves, the whole operation is controlled by the base-station (the eNodeB/eNB). Many crowded-platform scenarios fit in the above description. Examples include content-sharing applications in stadium, train stations and airports. In fact, in such scenarios, we have a high density of quasi-stationary UEs during the event or the waiting period. As a consequence, the need to relieve the macro-cell and micro/femto-cells is really vital.

The offloading operation entails both routing and OFDMA Resource Block (RB) allocation of the D2D communications over the SideLinks (SLs) interfaces of UEs . Note that the SL makes use of the same hardware transceiver and frequency spectrum employed by the UpLink (UL) interface. Consequently, any UE cannot simultaneously communicate in both interfaces UL and SL. Moreover, the SL communication is half-duplex [1]. In other words, the UE cannot simultaneously send and receive on SL.

Our objective is to solve jointly the two sub-problems where the routing decision considers the i) available OFDMA RBs, ii) interferences, and iii) dynamic state of network. However, a practical LTE-D2D-based offloading should not be agnostic to the fact that UEs are usually battery-limited devices. Therefore, to increase utility and lifetime of the offloading sub-network, we formulate an energy-aware joint scheme, for the OFDMA RB allocation and the routing, as a 0-1 Integer Linear Problem (ILP). It is worth noting that our formulation includes realistic constraints related to the LTE-D2D such as the half-duplex of SLs, the contiguity of RB allocations, the total power consumption due to the i) the baseband processing, ii) the RF transmission/reception of the D2D sidelink interface. Solving ILP problems, in general, has been proven to be NP-hard and the optimal solution is unlikely to be found in polynomial time [2].

To cope with the potentially-exponential complexity, we propose a new scheme named Joint Routing and Resource Allocation Energy Efficient (`JRRA-EE`). Our proposal is a centralized strategy hosted in the eNodeB. This means that the D2D communications are under the supervision of the telecommunication operator. Indeed, only the data plane is offloaded and the control plane is still under the control of the operator. `JRRA-EE` is a two-stage scheme. First, a pre-routing stage is performed in order to reduce the space of solutions. Then, the ILP is solved making use of the Branch-and-Cut approach while considering only the reduced space of candidates. The LTE-D2D offloading mechanism aims to help the eNodeB under heavy traffic conditions. Note that our proposed offloading mechanism is presented as a complementary method to deliver multicast and/or unicast flows. In other words, the purpose of LTE-D2D offloading is to complement, not to compete with, the LTE macro/femto-cells conventional delivery methods. If the LTE-D2D path does not exist, the communication will be ensured over the traditional macro/femto-cells. To gauge the effectiveness of `JRRA-EE`, we implemented the whole LTE-D2D SL protocol stack in UEs in the NS-3 network simulator. Using extensive simulations, we compared our proposal `JRRA-EE` to other one-sided optimal strategies and a non-energy aware variant. One-sided optimal strategy is one which is optimal only in one sense either in resource block allocation or in routing. Based on simulation results, we establish that our proposal `JRRA-EE` outperforms the other strategies in terms of i) the network lifetime, ii) the packet loss and iii) the service interruption

ISBN 978-3-903176-08-9

rate.

The remainder of the paper is organized as follows. Section II will summarize the related strategies addressing energy-aware multihop D2D communications. In Section III, we will formulate the problem. Then, in Section IV, the proposal will be detailed. The simulation environment and the performance evaluation will be presented in Section V. Finally, Section VI concludes the paper.

II. Related Work

In this section, we summarize the most relevant related strategies found in the literature dealing with energy-aware routing the context of D2D communications. Note that D2D is employed as an umbrella term for technologies that include, amongst others, LTE-D2D and WiFi Direct.

In [3], the authors show that LTE-D2D cooperative relays save significant amounts of energy when compared to conventional Base Station (BS) to UE communications. In addition, the authors present a cooperative relaying scheme to improve the UE's battery life. The idea consists in maximizing of use of UEs with high battery levels to deliver the traffic of UEs with low energy. Numerical simulations show that the approach reduces the outage probability of the cellular cooperating UEs.

In [4], the authors present a scheme to deliver BS-to-UE video content delivery via a cooperative D2D multihop routing. The proposed scheme employs a generic framework to avoid disruption caused by the depletion of D2D UE's energy budget. Seeking to optimize the budget utility, the algorithm described jointly schedules the routes and traffic workloads depending on the energy efficiency of each D2D wireless link.

In [5], the authors design an energy efficient routing protocol in Wi-Fi Direct cluster-based networks. The designed protocol adopts ideas from `LEACH` and `HEED` protocols which are well-known in wireless sensor networks. Through numerical simulations, the authors demonstrate that the scheme considerably saves network's energy as compared to the conventional peer-to-peer mode of Wi-Fi Direct.

In [6], the authors propose a heuristic algorithm energy-efficient multi-hop routing algorithm for UE-UE unicast traffic. Both channel reusing and power allocation are jointly addressed to obtain a satisfactory solution. The simulations show that significantly improvements in energy-efficiency of the multi-hop D2D communication system.

In relation to our paper, [3] considers only UE-to-BS traffic where high-battery UEs help low-battery ones to relay their traffic to the BS. On the other hand, our work focuses on offloading UE-UE multicast and/or unicast traffic to alleviate the base-station. Similarly, [4] also tackles the BS-to-UEs multicast video traffic where UEs employ a distributed multi-path routing and caching technique. In comparison, despite being energy-budget aware like [4], our work focuses on central algorithms and flow-centric applications where the employed on-demand cluster formation and caching, in [4], cannot be used. In the same manner, the protocol in [5] cannot be adapted into LTE-D2D to serve our purpose, since the traffic model in WSN is multiple-sources-one-sink and the clustering technique is useless in our case. The closest work to ours is [6] despite its focus on unicasting UE-UE traffic. However, the authors assume generic assumptions about wireless technology where the medium is abstracted as whole channels not in terms of resource blocks. They also consider an analytical power consumption model for each D2D link while no energy-budget limitation is considered. On the other hand, we consider a model of multicasting UE-to-UEs traffic where the unicast model can be treated as a special case. Furthermore, we incorporate LTE-D2D specificities with an empirical power consumption model to cater for the efficient utilization of the allocated energy budget for the cooperative relaying process.

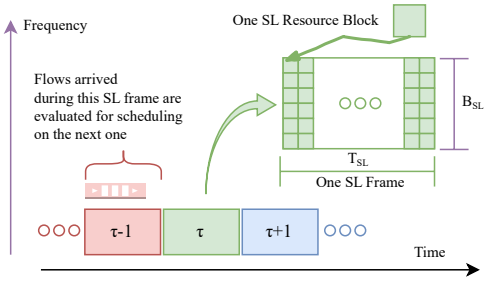


Fig. 1. Sidelink frame structure and scheduling.

III. Network and Problem Formulation

In this section, we will first provide a detailed model of our LTE-D2D system. Then, we will formulate our energy-aware routing and resource block allocation problem in LTE-D2D offloading networks.

A. System model

We consider N LTE-D2D enabled UEs who are located inside the LTE-A eNB's macro-cell. UEs are assumed to be quasi-stationary (e.g., located in the stadium) and are willing to offload the data plane of only **intra**-cell D2D traffic when it is expedient. The control plane is deployed in the eNB. The latter handles the offloading operations, over the D2D subnetwork, by continuously allocating OFDMA resource blocks during each sidelink (i.e., SL) frame. As depicted in Fig. 1, these operations are triggered each instant T where:

$$T = \tau \times T_{SL} \qquad \forall \tau \in \mathbb{N}$$

where T_{SL} and τ represent the duration of a SL frame and the frame index respectively. It is worth noting that the SL frame corresponds to the scheduling time unit in SL, which spans multiple one-millisecond time slots (i.e., multiple TTIs). Besides, it is characterized by a B_{SL} which corresponds to the total bandwidth of the SL communication, and is composed of Ω contiguous OFDMA resource blocks.

We model the D2D network as a symmetric directed graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$. Each node $v_i \in \mathcal{V}$ corresponds to one UE. An edge (i.e., sidelink) $e_{ij} \in \mathcal{E}$ between two nodes v_i and v_j exists if and only if the Signal to Noise Ratio (SNR), γ_{ij}, is greater than a predefined threshold, γ_{TOPO}. Formally,

$$\gamma_{ij} = \frac{g_{ij} P_{t,i}}{P_\sigma} \geq \gamma_{\text{TOPO}} \tag{1}$$

where i) $P_{t,i}$ is the power emitted by v_i, ii) P_σ corresponds to the thermal noise power, and iii) g_{ij} is the channel gain between the pair v_i and v_j which depends on the used channel model.

During each SL frame, the system's flows set, denoted by $\mathcal{F}$, is the union of two subsets $\mathcal{F}_{\text{S}}$ and $\mathcal{F}_{\text{C}}$ defined as follows:

- $\mathcal{F}_{\text{S}}$: set of *scheduled flows*. It corresponds to the on-going flows which are circulating on the LTE-D2D system. Hence, RBs need to be allocated for them in order to maintain their offloading operation.
- $\mathcal{F}_{\text{C}}$: set of *candidate flows*. It encompasses the flows which are waiting to be admitted during the next SL frame. Candidate flows are dynamically selected among those residing in the waiting queue, $\mathcal{F}_{\text{W}}$, based on the current availability of idle nodes $\mathcal{V}_{\text{D}}$.

Each multicast flow $f^k \in \mathcal{F}$ is characterized by a source node $s^k \in \mathcal{V}$, a destination group $\mathcal{D}^k \subseteq \mathcal{V}$ and a constant bit rate, R^k. Note that the unicast flow is a special case of the multicast flow in which $|\mathcal{D}^k| = 1$ (i.e., one destination). In this paper, we address only the Constant Bit-Rate (CBR)

TABLE I
NOTATION - SYSTEM MODEL

Symbol(s)	Meaning
v_n, e_{ij}, f^k	The n$^{\text{th}}$ node, the link from the i$^{\text{th}}$ to the j$^{\text{th}}$ node and the k$^{\text{th}}$ flow
$s^k, \mathcal{D}^k, \mathbb{T}^k$	The source, the destinations and the routing tree of f^k
R^k, D^k	The bit rate and the requested number of RBs of f^k
δ_x^y	Kronecker delta function which equals to 1 only when $x = y$ and 0 otherwise.
$\mathbb{1}_x^Y$	Set Y's indicator function which equals to 1 only when $x \in Y$ and 0 otherwise.
$\mathcal{O}(v_n), \mathcal{T}(v_n)$	Sets of outgoing (originating) links from v_n and terminating (incoming) links in v_n respectively
$x_{ij}^{h,k}$	Essential 0-1 decision variable that indicates whether the link e_{ij} is used to offload the flow f^k at the hop (tree level) number h
t_n^k	Auxiliary 0-1 variable indicating whether v_n acts as a (re-)transmitter for f^k. Note that at the source node, $t_{s^k}^k$ also indicates whether the flow is admitted or not.
H_n	Essential 0-1 decision variable that indicates the node v_n is scheduled to transmit during SL frames whose $p = \tau \bmod 2 = H_n$
$y_{u,n}$	Essential 0-1 decision variable that indicates that the RB pattern is allocated to the node v_n.
R_{ij}	Auxiliary 0-1 variable that indicates if the link e_{ij} is active.
R_n^ω	Auxiliary 0-1 variable indicates if the RB ω is allocated to v_n.
$R_n^{\omega,p}$	Auxiliary 0-1 variable indicates if the RB ω is allocated to v_n transmitting in the half duplex set p.
$R_{ij}^{\omega,p}$	Auxiliary 0-1 variable indicates if the RB ω is allocated to v_i transmitting to v_j in the half duplex set p.
$\phi_{n,ij}^{\omega,p}$	Auxiliary 0-1 variable indicates that v_n transmitting in the half duplex set p on the RB ω is interfering with the (active) link e_{ij}.

flows. Once admitted, f^k is carried throughout a routing tree $\mathbb{T}^k$ delivering its packets from s^k to $\mathcal{D}^k$. Note that if f^k is unicast flow, $\mathbb{T}^k$ is reduced to one branch (i.e., path). $\mathbb{T}^k$ is characterized by $h_{\max}$ levels which corresponds to the maximum number of hops from the root to the leaves.

Each UE is handled in an exclusive manner. This means that a given node can relay at most one flow at a time. Consequently, the routing trees (i.e., multicast) and/or branches (i.e., unicast) are mutually disjoint for concurrent flows.

B. Problem formulation

We address, in this paper, the energy-aware joint routing and OFDMA resource block allocation problem in LTE-D2D. The objective is to compute, for a given multicast flow $f^k \in \mathcal{F}$, the optimal routing tree while i) limiting the interferences between forwarding UEs, ii) minimizing the number of hops, iii) minimizing the communication energy consumption.

Conceptually, this problem can be decomposed into two sub-problems: i) routing and ii) resource allocation. However, following a cross-layer design, we propose to coordinate the resolution of the two sub-problems. In doing so, we aim to maximize the QoS and the efficiency of the system respectively from the point view of end-users and telecommunication operator. By such a joint treatment, enhanced results are obtained since the routing solution takes also in consideration, the induced wireless interferences in OFDMA RBs and the energy consumed in the transmission/reception operations.

In light of our adopted formalism shown in TABLE I, the routing problem can be formulated as following. Let $x_{ij}^{h,k}$ indicates whether the corresponding link e_{ij} is selected or not to be a part of a routing tree $\mathbb{T}^k$ for the flow f^k at the tree level (hop) h for $h = 0, 1, \ldots, h_{\max}$. To ensure a consistent tree structure, we introduce the following constraint which stipulates that a node v_n has at most one parent:

$$\sum_{\substack{e_{ij} \in \mathcal{T}(v_n) \\ 0 \le h \le h_{\max}, f^k \in \mathcal{F}}} \sum x_{ij}^{h,k} \le 1 \qquad \forall v_n \in \mathcal{V} \tag{2}$$

Note that $\mathcal{T}(v_n)$ corresponds to the set of incoming links to v_n. It is straightforward to see that this constraint ensures that a link cannot appear in more than one flow at a time.

Besides, we must ensure that only outgoing links from source are allowed at a tree's root (i.e., at $h = 0$). Formally,

$$x_{ij}^{h,k} \le \delta_0^h \cdot \delta_{v_i}^{s^k} + (1 - \delta_0^h)(1 - \delta_{v_i}^{s^k} - \delta_{v_j}^{s^k}) \quad \begin{matrix} \forall e_{ij} \in \mathcal{E} \\ \forall 0 \le h \le h_{\max} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{3}$$

We recall that δ_y^x corresponds to the Kronecker delta function which equals to 1 only when $x = y$ and 0 otherwise.

Also, we must guarantee that an outgoing link in the tree from a node v_n is possible at the level h if and only if an incoming link exists at the level $h - 1$. Formally,

$$x_{nm}^{h,k} \le \sum_{e_{ij} \in \mathcal{T}(v_n)} x_{ij}^{h-1,k} \qquad \begin{matrix} \forall e_{nm} \in \mathcal{E} \\ \forall 1 \le h \le h_{\max} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{4}$$

It is worth noting that constraints (2)–(4), also imply that a tree is a non-circular graph.

Besides, to prevent the addition of a needless branch stopping at a non-destination node, we require that only destination nodes are possible as leaves in a tree. Formally:

$$\sum_{\substack{e_{ij} \in \mathcal{T}(v_n) \\ 0 \le h \le h_{\max}}} \sum x_{ij}^{h,k} - \sum_{\substack{e_{ij} \in \mathcal{O}(v_n) \\ 0 \le h \le h_{\max}}} \sum x_{ij}^{h,k} \le \mathbb{1}_{v_n}^{\mathcal{D}^k} \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{5}$$

where $\mathbb{1}_x^Y$ corresponds to a set Y's indicator function, which equals to 1 only when $x \in Y$ and 0 otherwise, and $\mathcal{O}(v_n), \mathcal{T}(v_n)$ are the sets of outgoing links from v_n and terminating links in v_n respectively.

To ensure that the tree is formed only when it provides a complete delivery to all destinations, we add the following constraints:

$$\sum_{\substack{e_{ij} \in \mathcal{T}(v_n) \\ 0 \le h \le h_{\max}}} \sum x_{ij}^{h,k} \ge \mathbb{1}_{v_n}^{\mathcal{D}^k} \cdot t_{s^k}^k \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{6}$$

$$t_n^k \ge \sum_{0 \le h \le h_{\max}} x_{nm}^{h,k} \qquad \begin{matrix} \forall e_{nm} \in \mathcal{E} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{7}$$

$$t_n^k \le \sum_{\substack{e_{ij} \in \mathcal{O}(v_n) \\ 0 \le h \le h_{\max}}} \sum x_{ij}^{h,k} \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{8}$$

where t_n^k is a 0-1 auxiliary variable which is fixed by constraints (7) and (8) to indicate whether v_n acts as a relay node (i.e., a non-leaf node for flow f^k).

In addition, to ensure that the concurrently-admitted flows have non overlapping relay nodes, a node v_n is required to transmit at most one flow. Formally,

$$\sum_{f^k \in \mathcal{F}} t_n^k \le 1 \qquad \forall v_n \in \mathcal{V} \tag{9}$$

However, a node v_n may act, at once, as a source and a destination for two distinct flows. This case is not permitted. Formally,

$$\sum_{f^k \in \mathcal{F}} \left(\delta_{v_n}^{s^k} + \mathbb{1}_{v_n}^{\mathcal{D}^k} \right) \cdot t_{s^k}^k \le 1 \qquad \forall v_n \in \mathcal{V} \tag{10}$$

Fig. 2 illustrates an example of a routing tree generated according to the above constraints.

In line with the LTE-D2D standard [1], UEs are characterized by *half-duplex* D2D transmission in the side-link

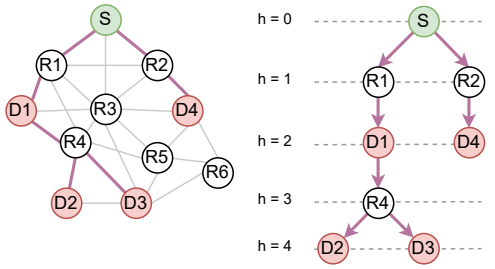


Fig. 2. Example of a constructed routing tree.

interface. Therefore, during a given SL, a node can act as a transmitter or a receiver but not both simultaneously. To cope with this hardware limitation, while reducing total end-to-end delay, we propose to schedule active links in an alternating fashion. To do so, we divide the set of active nodes $\mathcal{V}_\text{G} \subseteq \mathcal{V}$ into two *half-duplex sets*: $\mathcal{V}_\text{G}^0$ and $\mathcal{V}_\text{G}^1$. During a SL frame, the eNB scheduler addresses the RB allocations of one given set $\mathcal{V}_\text{G}^p$ depending on the parity p of the frame index τ (i.e., $p = \tau \mod 2$). Nodes belonging to the second set act as receivers. During the next frame, half-duplex sets switch their roles. As a result of this strategy, nodes in routing trees are scheduled according to the parity of their hop index in the routing tree (i.e., tree level). In other words, a parent node must belong to a different half-duplex set than its children. The node half-duplex allocation decision is embodied by the following constraint using the notations defined in TABLE I:

$$\sum_{\substack{f^k \in \mathcal{F} \\ 0 \le h \le h_\text{max}}} \sum x_{ij}^{h,k} \le H_i + H_j \le 2 - \sum_{\substack{f^k \in \mathcal{F} \\ 0 \le h \le h_\text{max}}} \sum x_{ij}^{h,k} \quad \forall e_{ij} \in \mathcal{E} \tag{11}$$

We assign a bandwidth B_SL, composed of Ω contiguous RBs, to the SL operation. Note that only contiguous RB allocations are feasible within this bandwidth because the SL has the same communication proprieties as the UL [7]. To do that, we enumerate all these allocations in the SL using a matrix $\mathcal{Z}_{\Omega \times U} = [z_{\omega,u}]$ in which columns represent the whole set contiguous patterns. The number of columns is given as $U = \frac{1}{2}\Omega(\Omega+1)$. Fo instance, all contiguous allocations for $\Omega = 4$ RBs are listed as columns in the following matrix:

$$\mathcal{Z}_{4\times 10} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

For example: the seventh column represents two RBs are allocated, namely the 3rd and 4th ones.

Furthermore, we model the RB allocation decision, for an active node v_n, as a set of respective 0-1 variables $y_{u,n} \forall u \in U$ which indicates the selected allocated pattern (i.e., column) u of the matrix $\mathcal{Z}$. This decision is constrained by:

$$\sum_{u=1}^{U} y_{u,n} \le \sum_{f^k \in \mathcal{F}} t_n^k \qquad \forall v_n \in \mathcal{V} \tag{12}$$

stipulating that only one allocation is possible for a node when it is acting as a (re-)transmitter for a flow.

To continue the formulation, additional auxiliary variables, whose definition are in TABLE I, are derived from the decision variables H_n and $y_{u,n}$ as detailed hereafter.

To indicate whether a RB ω is used by v_n, we introduce the 0-1 variable R_n^ω whose value is deduced from the respective variables $y_{u,n}$ by the following constraint:

$$R_n^\omega \triangleq \sum_{u=1}^{U} y_{u,n} z_{\omega,u} \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall 1 \le \omega \le \Omega \end{matrix} \tag{13}$$

Furthermore, additional variables $R_n^{\omega,0}$ and $R_n^{\omega,1}$ are defined to indicate whether the RB ω is used by v_n in $\mathcal{V}_\text{G}^0$ or $\mathcal{V}_\text{G}^1$, i.e. half-duplex set of frames, respectively. Formally,

$$R_n^{\omega,0} \triangleq R_n^\omega - R_n^{\omega,1} \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall 1 \le \omega \le \Omega \end{matrix} \tag{14}$$

$$R_n^{\omega,1} \triangleq H_n \cdot R_n^\omega \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall 1 \le \omega \le \Omega \end{matrix} \tag{15}$$

B_n is the number of RB used by v_n and it is equal to:

$$B_n \triangleq \sum_{\omega=1}^{\Omega} R_n^\omega \qquad \forall v_n \in \mathcal{V} \tag{16}$$

An additional set of link-level auxiliary 0-1 variables are introduced as follows:

$$R_{ij} \triangleq \sum_{\substack{0 \le h \le h_\text{max} \\ f^k \in \mathcal{F}}} \sum x_{ij}^{h,k} \qquad \forall e_{ij} \in \mathcal{E} \tag{17}$$

$$R_{ij}^{\omega,p} \triangleq R_i^{\omega,p} \cdot R_{ij} \qquad \begin{matrix} \forall e_{ij} \in \mathcal{E} \\ \forall 1 \le \omega \le \Omega \\ \forall p \in \{0,1\} \end{matrix} \tag{18}$$

$$\phi_{n,ij}^{\omega,p} \triangleq R_n^{\omega,p} \cdot R_{ij}^{\omega,p} \qquad \begin{matrix} \forall v_n \in \mathcal{V}, \forall e_{ij} \in \mathcal{E} \\ \forall 1 \le \omega \le \Omega \\ \forall p \in \{0,1\} \end{matrix} \tag{19}$$

where R_{ij} indicates if e_{ij} is used for some flow. $R_{ij}^{\omega,p}$ indicates if the RB ω is used for the scheduled link e_{ij} during the p^th half duplex set. $\phi_{n,ij}^{\omega,p}$ is an interference indicator between node v_n and link e_{ij} on the RB ω.

To adhere to a linear formulation, a further step is needed to linearize Constraints (15), (18) and (19), which contain product terms. We make use of standard technique by introducing for each term $x \cdot y$ an additional auxiliary 0-1 variable λ_{xy} add three more linear constraints as follows:

$$(\lambda_{xy} \le x) \wedge (\lambda_{xy} \le y) \wedge (\lambda_{xy} \ge x + y - 1) \tag{20}$$

To increase the RB reutilization and reduce power consumption, we require that UEs cannot allocate RBs more than the flows' requests. Formally,

$$B_n \le \Omega + (D^k - \Omega) \cdot t_n^k \qquad \begin{matrix} \forall v_n \in \mathcal{V} \\ \forall f^k \in \mathcal{F} \end{matrix} \tag{21}$$

Note that the relation between flow bit-rate R^k and the respective demand for RBs D^k is defined by [7] as:

$$R^k = \frac{\text{TBS}\left(\text{MCS}, D^k\right)}{C} \qquad \text{[Mbps]} \tag{22}$$

where TBS is the MAC transport block size function in bits as defined in [7] considering a baseline *modulation and coding scheme* (MCS) for the SL. C is a constant equal to 1000.

In our model, we adopt a fixed power density scheme for the D2D emission power. In this scheme, the total emission power $S_{\text{tx},n}$ of a node is proportional to the number of allocated RBs B_n. Formally,

$$S_{\text{tx},n} = \Psi_{t,n} \cdot B_n \qquad \text{[mW]} \tag{23}$$

Furthermore, we assume a common emission power density for the D2D nodes (i.e., $\Psi_{t,n} = \Psi_t$, $\forall v_n \in \mathcal{V}$).

Following the same *per-RB* treatment and assuming *flat block-fading* channel model, the overall Signal-to-Interference-plus-Noise Ratio (SINR) on the link e_{ij} is equal to:

$$\gamma_{ij} = \frac{g_{ij}\Psi_{t,i}}{\sum\limits_{v_n \in \mathcal{V}} g_{nj}\Psi_{t,n} + \Psi_\sigma} \tag{24}$$

where Ψ_σ and $\Psi_{t,n}$ represent the spectral densities (per RB) of the thermal noise and the transmission from v_n, and g_{ij} is the channel gain between the node pair (v_i, v_j).

In face of the reutilization of RBs, system performance is limited by interference caused by nodes transmitting using the same RB. To optimize the performances by minimizing interferences, SINR must be upper-bounded by a common threshold γ. To formulate this constraint on RB allocations, we translate this limit (i.e., SINR $\le \gamma$) into the inequality

TABLE II
UE POWER CONSUMPTION MODEL PARAMETERS.

Parameter	Value	Parameter	Value
$P_{\text{tx}}^{\text{const}}$	883.52 mW	a_1^{rx}	24.8 mW
$P_{\text{rx}}^{\text{const}}$	878.1 mW	a_2^{rx}	7.86 mW
s_1^{tx}	0.2 dBm	a^{R}	8.16 mW
s_2^{tx}	11.4 dBm	b^{R}	0.97 mW/Mbps
s_1^{rx}	52.5 dBm	b_1^{tx}	0.78 mW/dBm
a_1^{tx}	23.6 mW	b_2^{tx}	17 mW/dBm
a_2^{tx}	45.4 mW	b_1^{rx}	0.04 mW/dBm
		b_2^{rx}	0.11 mW/dBm

$\mathcal{N}+\mathcal{I} \leq P_r/\gamma$ where P_r is the received power. Consequently,

$$\Psi_\sigma R_{ij}^{\omega,p} + \sum_{n\neq i} g_{nj}\Psi_t \cdot \phi_{n,ij}^{\omega,p} \leq \frac{g_{ij}\Psi_t}{\gamma} R_{ij}^{\omega,p} \quad \begin{matrix}\forall e_{ij}\in\mathcal{E} \\ \forall 1\leq\omega\leq\Omega \\ \forall p\in\{0,1\}\end{matrix} \quad (25)$$

where constraint (25) ensures that the SINR is below the threshold γ considering RB allocations and active nodes interfering in the same half-duplex set $\mathcal{V}_{\text{G}}^p$. The auxiliary 0-1 variable $R_{ij}^{\omega,p}$ indicates that the link e_{ij} is scheduled to transmit together with half-duplex set $\mathcal{V}_{\text{G}}^p$ on the RB ω.

To evaluate the effect of energy consumption, similar to [8], we make use of the following empirical model defined in [9], to calculate the total communication consumed power due to the D2D operations at both ends. At the transmitting end, the total consumed power is given by:

$$P_{\text{tx}}^{\text{D2D}} = P_{\text{tx}}^{\text{const}} + P_{\text{tx}}^{\text{RF}}(S_{\text{tx}}) \quad (26)$$

$$P_{\text{tx}}^{\text{RF}}(S_{\text{tx}}) = \begin{cases} b_1^{\text{tx}} \cdot S_{\text{tx}} + a_1^{\text{tx}} & \text{if } S_{\text{tx}} \leq s_1^{\text{tx}} \\ b_2^{\text{tx}} \cdot S_{\text{tx}} + a_2^{\text{tx}} & \text{if } s_1^{\text{tx}} < S_{\text{tx}} \leq s_2^{\text{tx}} \end{cases}$$

where $P_{\text{tx}}^{\text{D2D}}$ includes the constant term $P_{\text{tx}}^{\text{const}}$ related to the baseband circuit consumption when the D2D transmitter is active. $P_{\text{tx}}^{\text{RF}}$ is the total RF block consumption in terms of the power emitted S_{tx} from the antenna in dBm. Hence, the power consumed by a node v_n, due to the transmission of the flow f^k, can be estimated as:

$$\Pi_{\text{tx},n}^k = P_{\text{tx}}^{\text{const}} + P_{\text{tx}}^{\text{RF}}\left(S_{\text{tx,n}}^k\right) \quad [\text{mW}] \quad (27)$$

$$S_{\text{tx,n}}^k = \text{dBm}\left(\Psi_{t,n} D^k\right) \quad (28)$$

Similarly, at the receiving end of an active link, the total consumed power is equal to:

$$P_{\text{rx}}^{\text{D2D}} = P_{\text{rx}}^{\text{const}} + P_{\text{rx}}^{\text{RF}}(S_{\text{rx}}) + P_{\text{rx}}^{\text{BB}}(R) \quad (29)$$

$$P_{\text{rx}}^{\text{RF}}(S_{\text{rx}}) = \begin{cases} -b_1^{\text{rx}} \cdot S_{\text{rx}} + a_1^{\text{rx}} & \text{if } S_{\text{rx}} \leq -s_1^{\text{rx}} \\ -b_2^{\text{rx}} \cdot S_{\text{rx}} + a_2^{\text{rx}} & \text{if } S_{\text{rx}} > -s_1^{\text{rx}} \end{cases}$$

$$P_{\text{rx}}^{\text{BB}}(R) = b^{\text{R}} \cdot R + a^{\text{R}}$$

where $P_{\text{rx}}^{\text{D2D}}$ includes the constant term $P_{\text{rx}}^{\text{const}}$, related to the receiving circuit being active, and $P_{\text{rx}}^{\text{RF}}$, which gives the total RF block consumption in terms of the power received S_{rx} at the antenna in dBm. The additional term $P_{\text{rx}}^{\text{BB}}$ gives the rate-dependent power consumption in the base-band block of the device. Therefore, the power consumption at the receiver of an active link e_{ij}, due to the reception of the flow f^k, is estimated by:

$$\Pi_{\text{rx},ij}^k = P_{\text{rx}}^{\text{const}} + P_{\text{rx}}^{\text{RF}}\left(S_{\text{rx},ij}^k\right) + P_{\text{rx}}^{\text{BB}}\left(R^k\right) \quad [\text{mW}] \quad (30)$$

$$S_{\text{rx},ij}^k = \text{dBm}\left(g_{ij} \cdot \Psi_{t,i} D^k\right) \quad (31)$$

where R^k is the respective flow bit-rate defined in equation (22).

TABLE II illustrates the parameters values of the above power consumption model.

We model the impact of node participation in routing on its residual energy by proposing a ranking method that takes into consideration the current distribution of residual energy in the system. For each non-dead node v_n, we assign a fractional rank $\Lambda_n \in (0, 1]$ as following:

$$\Lambda_n(\tau) = \frac{1}{1 + \left\lfloor \frac{E_n(\tau) - E_{\min}(\tau)}{\sigma_E(\tau)} \right\rfloor} \quad (32)$$

where $E_n(\tau)$, $E_{\min}(\tau)$ and $\sigma_E(\tau)$ respectively represents i) node's residual energy, ii) minimum residual energy in the network, and iii) standard deviation of residual energy distribution at the beginning of the SL frame τ. Note that high fractional rank means high impact on the node's residual energy.

The time evolution of the residual energy is estimated at the eNodeB as detailed hereafter:

$$E_n(\tau) = E_n(\tau - 1) - P^{\text{D2D}} \cdot T_{\text{SL}} \quad (33)$$

$$P^{\text{D2D}} = \begin{cases} P_{\text{tx}}^{\text{D2D}} \text{ if } v_n \text{ was transmitting in frame } \tau - 1 \\ P_{\text{rx}}^{\text{D2D}} \text{ if } v_n \text{ was receiving in frame } \tau - 1 \\ 0 \quad \text{if } v_n \text{ was idle in frame } \tau - 1 \end{cases}$$

assuming that each node has initial energy budget $E_n(0)$.

Above, we have defined all the variables and constraints (2) – (19) addressing i) routing, ii) OFDMA RB allocation and iii) energy consumption. Now, we can complete the formulation by defining the objective function:

$$\max_{x_{ij}^{h,k}, H_n, \ldots} \quad \frac{1}{\aleph_{\text{B}}} \sum_{v_n\in\mathcal{V}} B_n + \frac{1}{\aleph_{\text{A}}} \sum_{f^k\in\mathcal{F}} t_{s^k}^k - \frac{1}{\aleph_{\text{R}}} \sum_{\substack{v_n\in\mathcal{V} \\ f^k\in\mathcal{F}}} \sum \Lambda_n t_n^k$$
$$- \frac{1}{\aleph_{\text{tx}}} \sum_{\substack{v_n\in\mathcal{V} \\ f^k\in\mathcal{F}}} \sum \Pi_{\text{tx},n}^k t_n^k - \frac{1}{\aleph_{\text{rx}}} \sum_{e_{ij}\in\mathcal{E}} \sum_{0\leq h\leq h_{\max}, f^k\in\mathcal{F}} \sum \Pi_{\text{rx},ij}^k x_{ij}^{h,k} \quad (34)$$

where the normalizing factors defined by:

$$\aleph_{\text{B}} \triangleq \Omega \cdot |\mathcal{V}|, \aleph_{\text{A}} \triangleq |\mathcal{F}_{\text{C}}|, \aleph_{\text{R}} \triangleq \sum_{v_n\in\mathcal{V}} \Lambda_n,$$
$$\aleph_{\text{tx}} \triangleq \sum_{\substack{v_n\in\mathcal{V} \\ f^k\in\mathcal{F}}} \sum \Pi_{\text{tx},n}^k, \aleph_{\text{rx}} \triangleq \sum_{e_{ij}\in\mathcal{E}, f^k\in\mathcal{F}} \sum \Pi_{\text{rx},ij}^k \quad (35)$$

The terms in the objective function in (34), represent *respectively* a normalized equal-weight multi-objective formulation of eNodeB goal to achieve the following objectives: i) increasing the number of RB allocated for each flow, ii) increasing the number of admitted flows in the system, iii) lowering the routing impact on nodes' residual energy, iv) lowering the power consumption in the relaying process at the transmitting side, v) lowering the power consumption in the relaying process at the receiving side.

To quantify the ILP model's size complexity, we cite its *column-size*, i.e., number of variables, and its *row-size*, i.e., number of constraints. As for our model, an asymptotic analysis shows that, in terms of $\mathbb{G}$, $\mathcal{F}$, Ω and $h_{\max}$, the ILP model has column-size of $\mathcal{O}\left(|\mathcal{V}|\,\Omega^2 + |\mathcal{V}|\,|\mathcal{E}|\,\Omega + |\mathcal{V}|\,|\mathcal{F}| + |\mathcal{E}|\,|\mathcal{F}|\,h_{\max}\right)$ and a row-size of $\mathcal{O}\left(|\mathcal{V}|\,|\mathcal{E}|\,\Omega + |\mathcal{V}|\,|\mathcal{F}| + |\mathcal{E}|\,|\mathcal{F}|\,h_{\max}\right)$.

IV. PROPOSAL: JRRA−EE

To solve our energy-aware joint routing and OFDMA resource block allocation problem, formulated in the above section as an ILP model, we propose a two-stage heuristic algorithm, based on the branch-and-cut method, named Joint Routing and Resource Allocation Energy Efficient (JRRA−EE). It is worth noting that our scheme is centralized and is handled by the eNodeB.

JRRA−EE adopts an online bulk strategy by considering for the SL frame τ resolution, all active scheduled flows and the waiting flows up to the previous SL frame. However, instead of considering all the waiting flows $\mathcal{F}_{\text{W}}$ for admittance, it proceeds by an initial stage of pre-routing to filter the waiting

Algorithm 1 JRRA−EE pseudo-code

for each SL frame τ **do**
  for each $f^k \in \mathcal{F}_{\text{A}}$ **do** ▷ Arriving Flows
    $\mathcal{F}_{\text{W}} \leftarrow \mathcal{F}_{\text{W}} \cup \{ f^k \}$
  end for
  for each $f^k \in \mathcal{F}_{\text{FIN}}$ **do** ▷ Finished Flows
    $\mathcal{V}_{\text{D}} \leftarrow \mathcal{V}_{\text{D}} \cup \text{NodesOF}\left(\mathbb{T}^k\right)$
  end for
  Execute Algorithm 2 ▷ Pre-routing
  Construct the ILP model as in formula (34)
  Solve the ILP model using branch-and-cut
  for each $f^k \in \mathcal{F}_{\text{C}}$ **do**
    if $t^k_{s_k} = 1$ **then** ▷ Flow is admitted
      Configure $\mathbb{T}^k$ according to $x^{h,k}_{ij}$
    end if
  end for
  $p \leftarrow \tau \mod 2$
  for each $v_n \in \mathcal{V}^p_{\text{G}}$ **do**
    Allocate RBs for v_n according to $y_{u,n}$
  end for
end for

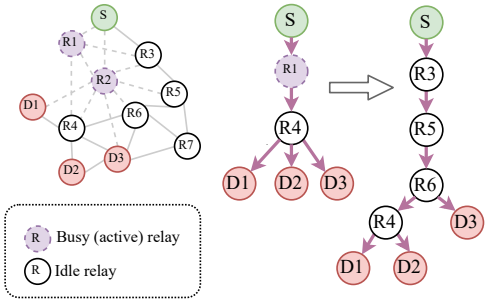


Fig. 3. Pre-routing tree formation and its deviation.

flows down to a set of candidate flows $\mathcal{F}_{\text{C}}$. The rational behind this initial stage is to reduce the size complexity of the ILP model by reducing the number of considered flows $\mathcal{F}$ and also by setting the model parameter $h_{\max}$ to a reasonable value. It is worth noting that high values for $h_{\max}$ implies more possible routing trees to discover while low values few routing trees and hence few admitted flow into the system. The pseudo-code of JRRA−EE illustrated in Algorithm 1.

The pre-routing stage proceeds as follows. For each waiting flow f^k, the eNodeB checks if it is possible to construct a routing tree from the source node to all destinations using *breadth-first-traversal* and considering only the currently idle nodes. We recall that each node cannot handle more than one flow. Note that such tree construction stops once all destinations are reached. If such *pre-routing* tree $\widetilde{\mathbb{T}}^k$ exists then the flow f^k is added to the set of candidate flows $\mathcal{F}_{\text{C}}$. In the other case, the flow is kept waiting for upcoming opportunities in subsequent frames. Thanks to the breadth-first-traversal, pre-routing trees are *well-balanced* as they tend to be short one-to-many routing trees. However, due to the dynamic state (e.g., end of current flows, low battery, etc.) of nodes, pre-routing trees also tend to deviate from this preferred condition as illustrated in Fig. 3. The pseudo-code of the pre-routing trees construction is illustrated in Algorithm 2.

Taking advantage of the dynamic nature of pre-routing trees construction stage, the latter goes one step further to set the parameter value $h_{\max}$ of the current ILP model based on the reported trees heights and those of the routing trees of active

Algorithm 2 Pre-routing of routing trees pseudo-code

Inputs: $\mathcal{V}_{\text{D}}, \mathbb{T}^k \, \forall f^k \in \mathcal{F}_{\text{S}}$, $\mathcal{F}_{\text{W}}$
Outputs: $\mathcal{F}_{\text{C}}, h_{\max}$

$\mathcal{F}_{\text{C}} \leftarrow \emptyset, h \leftarrow 0, \widetilde{h} \leftarrow 0$
for each $f^k \in \mathcal{F}_{\text{S}}$ **do** ▷ Trees of active flows
  if HeightOF $\left(\mathbb{T}^k\right) > h$ **then**
    $h \leftarrow$ HeightOF $\left(\mathbb{T}^k\right)$
  end if
end for
for each $f^k \in \mathcal{F}_{\text{W}}$ **do**
  if $\{v_{s^k}\} \cup \mathcal{D}^k \nsubseteq \mathcal{V}_{\text{D}}$ **then go to** 30
  end if
  $Q \leftarrow \emptyset$ ▷ New empty queue
  push v_{s^k} into Q
  $S \leftarrow \{v_{s^k}\}$
  LevelOF$(v_{s^k}) \leftarrow 0$
  while $Q \neq \emptyset \wedge \mathcal{D}^k \nsubseteq S$ **do** ▷ Breadth-first traversal
    $v_i \leftarrow$ Q.pop()
    if LevelOF$(v_i) > \widetilde{h}$ **then**
      $\widetilde{h} \leftarrow$ LevelOF(v_i)
    end if
    for each $e_{ij} \in \mathcal{O}(v_i)$ **do**
      if $v_j \notin S \wedge v_j \in \mathcal{V}_{\text{D}}$ **then**
        push v_j into Q
        $S \leftarrow S \cup \{ v_j \}$
        LevelOF$(v_j) \leftarrow$ LevelOF$(v_i) + 1$
      end if
    end for
  end while
  if $\mathcal{D}^k \subseteq S$ **then** ▷ Add f^k to candidates
    $\mathcal{F}_{\text{C}} \leftarrow \mathcal{F}_{\text{C}} \cup \{f^k\}$
  end if
end for
$h_{\max} \leftarrow \max\left\{h, \beta \cdot \widetilde{h}\right\}$ ▷ Update $h_{\max}$

scheduled flows as follows:

$$h_{\max} = \max\left\{ \max_{f^k \in \mathcal{F}_{\text{S}}} \mathfrak{H}\left(\mathbb{T}^k\right), \beta \max_{f^k \in \mathcal{F}_{\text{C}}} \mathfrak{H}\left(\widetilde{\mathbb{T}}^k\right) \right\} \quad (36)$$

where i) $\mathfrak{H}(\cdot)$ denotes the height-of-tree operator and ii) $\beta \geq 1$ is a "tradeoff-margin" factor to allow for longer routing trees to be explored and more flows to be admitted into the system when solving the current ILP model. After this initial stage, we drastically reduce the size of the solutions, hence the eNodeB can solve the resulted ILP model using the branch-and-cut method and the convergence time is tiny.

V. Performance Evaluation

In this section, we report the performance of our proposal JRRA−EE by performing a series of detailed simulations. We start by describing the network simulation environment setup. Afterwards, we define the performance metrics to evaluate our strategy. Finally, we analyze the results and discuss the effectiveness of our proposal JRRA−EE based on multiple-run simulations which invoke confidence-interval analysis with a confidence level of 95%.

A. Network simulation environment

We make use of NS-3 network simulator based on C++ language and widely used by the network research community. NS-3 supports a variety of conventional 3GPP LTE simulation scenarios through the module NS-3/LTE [10]. To realize the LTE-D2D standard, we integrate new features to NS-3 in order to support LTE-D2D protocol stack (side-link interface). In this context, we developed the necessary LTE-D2D procedures for the layers: PHY, MAC and PDCP/RLC. We also

TABLE III
SIMULATION PARAMETERS

Parameter	Value
Cell Radius Rcell	1 km
UL/SL Frequency f_{UL}	1930 MHz
UL/SL (Reference) Bandwidth B_{UL}	5 MHz (25 LTE RBs)
SL RBs Used Actually Ω	14 LTE RBs
SL frame (LTE-D2D SC-Period)	40 subframes (40 ms)
Data Part in SL frame	32 subframes
UE SL Power Transmit Density Ψ_t	-4 dBm/RB
Noise Spectral Density Ψ_n	-121.45 dBm/RB
LTE MCS Index used in SL	9 (QPSK)
UE Density λ_{UE}	$\{10, 15, 20, 25, 30, 35, 40\}$ per km^2
UE-UE SNR Threshold γ_{TOPO}	10 dB
Scheduling SINR Threshold γ	6 dB
UE Initial Energy Budget $E_n(0)$	3.856 Joules
Flow Simulation Period	10 seconds
Flow Arrival Process	Poisson Process
Flow Arrival Rates λ_{FL}	$\{10, 20\}$ flows/second
Flow Duration Random Variable	Exponential
Flow Duration Mean λ_{DUR}	1 second
Flow Bit Rate Classes	$\{25, 50, 75, 100, 125, 150, 175, 200\}$ kbps
Node-Flow Interest Probability ρ	0.1
$h_{\max}$ update factor β	1.5

implemented the signaling necessary to: i) configure the SL parameters, ii) establish SL Radio Bearers (SLRBs), and iii) exchange SL reports and grants.

In line with our formulation in Section III, we deploy one LTE macro-cell with radius Rcell $= 1$ km. The deployed UEs follow a Poisson Point Process distribution with a density λ_{UE} nodes per km^2 for values from $\{10, 15, 20, 25, 30, 35, 40\}$. The LTE macro-cell is configured to work with an UL/SL frequency of 1930 MHz (i.e., band 1) and a bandwidth of 5 MHz (i.e., 25 RBs). However, we assign only $\Omega = 14$ RBs for the actual SL bandwidth of D2D offloading operation. All UEs transmit on SL with a common power density of $\Psi_t = -4$dBm/RB which is equivalent to a maximum of 10 dBm over the whole 5 MHz. To model the SL path-loss (i.e., link gains g_{ij}), we make use of WINNER II B2-LOS channel model [11]. The SL frame duration is fixed to 40 milliseconds which corresponds to 40 LTE subframes. Note that only 32 subframes are actually used for data transmission while the initial 8 ones are used for SL control information. The eNodeB builds the D2D network topology making using of SNR reports and estimations (i.e., CQI metric). A communication link exists between two nodes if and only if the respective SNR is greater than a threshold $\gamma_{\text{TOPO}} = 10$ dB.

Simulated flows are generated following a Poisson process with an arrival rate equals to $\lambda_{\text{FL}} \in \{10, 20\}$ flows per second. Flow bit-rates are randomly selected from predefined Constant Bit Rate (CBR) classes. Flow duration distribution is simulated to follow an exponential random variable with a mean duration of $\lambda_{\text{DUR}} = 1$ second. Flows sources are selected according a random uniform distribution. As for destinations, they are selected for a given source assuming a *node-flow interest probability* of $\rho = 0.1$. In other words, once a flow source is selected, other nodes are evaluated for being interested in receiving the flow using Bernoulli trials with a success probability equals to ρ. TABLE III summarizes the main parameters used in simulations.

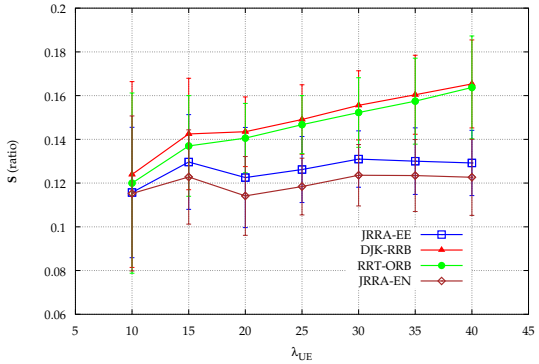
B. Performance metrics

As described in TABLE IV, we consider various metrics to evaluate purposes in our experiments. These metrics are grouped with respect to the following interests: i) I1: overall utility of offloading system, ii) I2: end-users' quality of service per flow, and iii) I3: energy consumption.

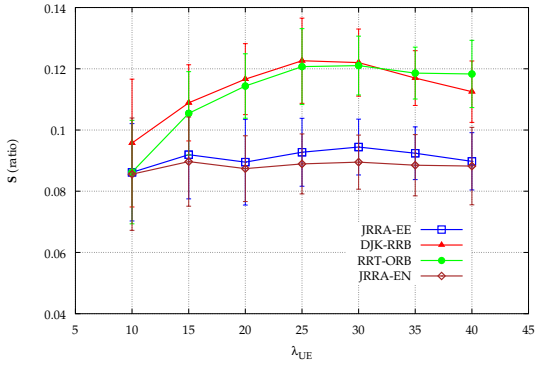
Unfortunately, the related strategies described in Section II cannot be compared with our proposal. The main reason

TABLE IV
PERFORMANCE METRICS

Metric	Definition	Interest
$\mathbb{S}$	ratio of the flows offloaded by the D2D subnetwork	I1
$\mathbb{I}$	ratio of interrupted flows (to the admitted ones) due to topology disruption by death of relays	I1,I3
$\mathbb{L}$	average of flow's packet loss in each simulation run	I2
$\mathbb{E}_n$	average network life time as n connected components	I3
$\mathbb{H}$	average height (hops) of trees in each simulation run	I2



(a) $\lambda_{\text{FL}} = 10$



(b) $\lambda_{\text{FL}} = 20$

Fig. 4. $\mathbb{S}$ w.r.t node density λ_{UE}.

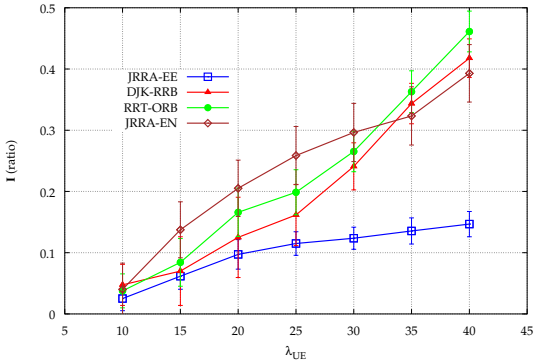
behind this is that the full optimization model considered in our paper (Section III) is not addressed at all in the related papers. Hence, we propose to compare JRRA-EE with the following variants:

1) DJK-RRB: is a pure path strategy that aims to find the optimal routing trees using the one-to-many version Dijkstra algorithm and then, allocates RB randomly.
2) RRT-ORB: is a pure resource block oriented strategy that finds the routing trees randomly using random walk on the topology graph, and allocates RB optimally.
3) JRRA-EN: is an energy non-aware variant of the original JRRA-EE. It relies, hence, on a modified objective function as described below:

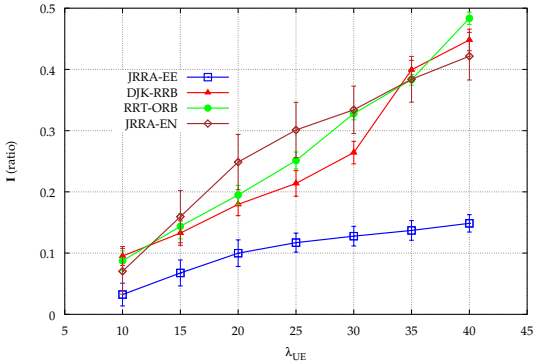
$$\max_{x_{ij}^{h,k}, H_n, \cdots} \quad \frac{1}{\aleph_{\text{B}}} \sum_{v_n \in \mathcal{V}} B_n + \frac{1}{\aleph_{\text{A}}} \sum_{f^k \in \mathcal{F}} t_{s^k}^k - \frac{1}{\aleph_{\text{N}}} \sum_{\substack{v_n \in \mathcal{V} \\ f^k \in \mathcal{F}}} \sum t_n^k$$

$$\text{with} \qquad \aleph_{\text{B}} \triangleq \Omega \cdot |\mathcal{V}|, \aleph_{\text{A}} \triangleq |\mathcal{F}_{\text{C}}|, \aleph_{\text{N}} \triangleq |\mathcal{V}| \qquad (37)$$

where the new normalized term $-\frac{1}{\aleph_{\text{N}}}[\cdots]$ represents the eNodeB's attempt to minimize the number of involved nodes in the offloading route.

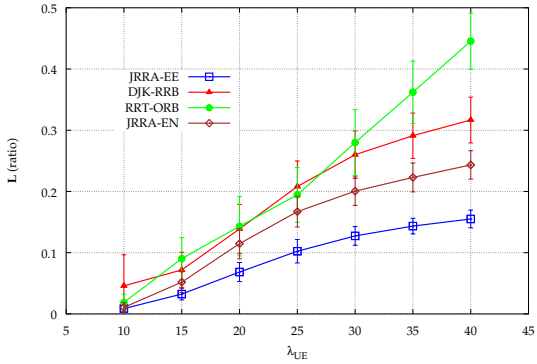


(a) $\lambda_{\text{FL}} = 10$

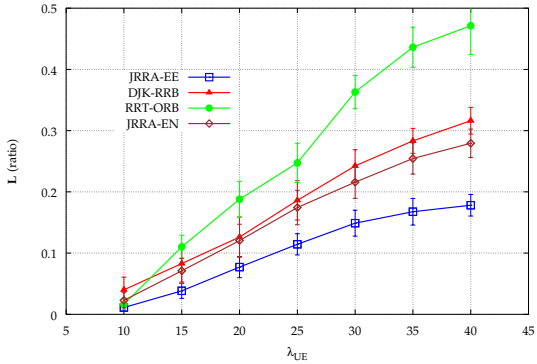


(b) $\lambda_{\text{FL}} = 20$

Fig. 5. $\mathbb{I}$ w.r.t node density λ_{UE}.



(a) $\lambda_{\text{FL}} = 10$



(b) $\lambda_{\text{FL}} = 20$

Fig. 6. $\mathbb{L}$ w.r.t node density λ_{UE}.

C. Simulation results

First, we evaluate the offloading capability of our approach compared with the related strategies. To do so, we measure the ratio of the flows offloaded by the D2D subnetwork. Fig. 4 illustrates $\mathbb{S}$ with respect to the density of UEs and under two traffic conditions $\lambda_{\text{FL}} = 10$ and $\lambda_{\text{FL}} = 20$ flows per second. We note that DJK−RRB generally outperforms the other strategies. This is expected since DJK−RRB routes flows over the fewest possible nodes (i.e., the smallest possible trees). Hence, it allows for more flows to be admitted. Taking DJK−RRB as a baseline, we note that our proposal JRRA−EE accepts, in average, around $\mathbb{S} = 12\%$ of the flows with $\lambda_{\text{FL}} = 10$ which is $\Delta\mathbb{S} = 4\%$ less than DJK−RRB as illustrated in Fig. 4a. On the other hand, Fig. 4b depicts $\mathbb{S}$'s variation under a higher traffic load. We notice that, for $\lambda_{\text{FL}} = 20$, the performance of JRRA−EE drops to around $\mathbb{S} = 9\%$. However, the most advantageous DJK−RRB also drops making the performance gap of JRRA−EE within $\Delta\mathbb{S} = 3\%$. It is straightforward to see that, even though JRRA−EE is outperformed by DJK−RRB and RRT−ORB, our proposal performs better than its energy non-aware variant JRRA−EN in terms of offloading capacity.

Fig. 5 depicts the ratio of interrupted flows according to the UEs' density for respectively $\lambda_{\text{FL}} = 10$ and $\lambda_{\text{FL}} = 20$ scenarios. From flows perspective, the admission rate (i.e., as illustrated in Fig. 4) alone is not sufficient and we have to ensure that the path is valid until the reception of all the packets. It is worth pointing out that our proposal achieves the lowest service interruption probability compared with the related strategies. Fig. 5a and Fig. 5b clearly demonstrate that JRRA−EE resists well to the traffic increase. In fact, it is able to maintain the service interruption rate $\mathbb{I}$ below 15% under both traffic conditions $\lambda_{\text{FL}} = 10$ and $\lambda_{\text{FL}} = 20$ flow per seconds. On the other hand, DJK−RRB which is able to maximize the offloading rate, struggles to resist to such an increase and may crash more than 40%.

Besides, to quantify the QoS in terms of packet error rate at the IP level, Fig. 6 illustrates the average flow's packet loss ($\mathbb{L}$) according to the UEs' density for respectively $\lambda_{\text{FL}} = 10$ and $\lambda_{\text{FL}} = 20$ scenarios. We recall that the packet loss does not come only from transmission error due to noise interference but may also be caused by the service interruption. Indeed, a flow may be disrupted because a relaying node has exhausted all its energy budget and consequently declared itself as dead. In Fig. 6, it is straightforward to see that JRRA−EE and JRRA−EN both outperform DJK−RRB and RRT−ORB thanks to their capability to take into consideration interference in OFDMA RB blocks allocation. However, RRT−ORB performs badly in general which may seems paradoxical. The rational behind this is RRT−ORB handles RBs allocation once the routes are randomly selected leaving few possibilities to allocate sufficient RBs to flows. It is straightforward to see that such a behaviour will lead to higher transmission delays. As a consequence, longer transmission delays paired with energy-agnostic node selection for routing is resulting in high packet loss due to the service disruption.
Being energy-aware makes JRRA−EE more robust against the packet loss. In fact, the latter succeeds to maintain $\mathbb{L}$ below 0.15 and 0.18 for both traffic conditions $\lambda_{\text{FL}} = 10$ and $\lambda_{\text{FL}} = 20$ flow per seconds respectively as depicted in Fig. 6a and Fig. 6b.

To highlight the energy efficiency of our proposal JRRA−EE, we make use of the metric $\mathbb{E}_n$ which measures the average lifetime of the D2D offloading system as a n connected components (i.e., evolution of network connectivity). Disruptions caused by nodes' energy shortage lead to the topology disconnection which, in its turn, degrades the overall utility of the D2D offloading system. In this regard, Fig. 7 highlights how JRRA−EE succeeds to keep the D2D topology

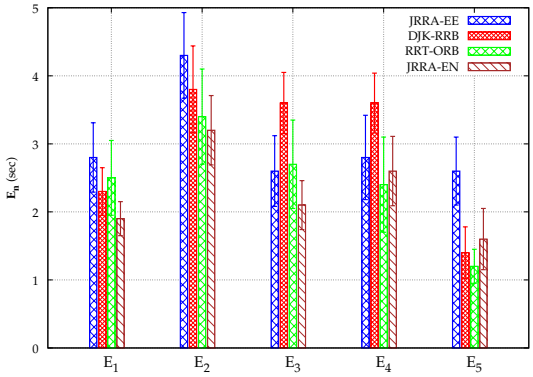
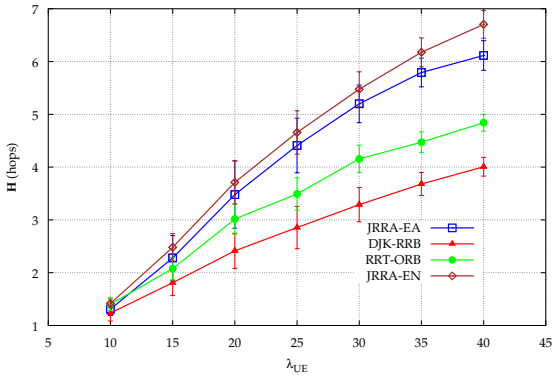
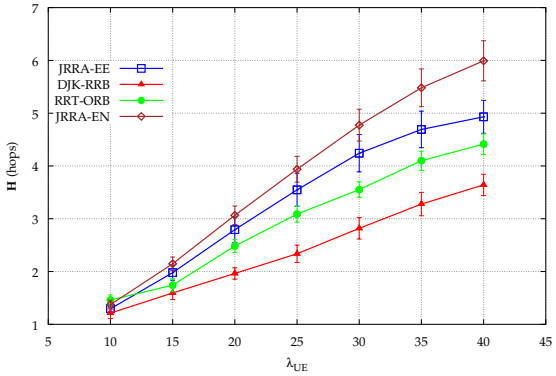


Fig. 7. Network connectivity lifetime $\mathbb{E}_n$ for $\lambda_{\text{FL}} = 20$.



(a) $\lambda_{\text{FL}} = 10$



(b) $\lambda_{\text{FL}} = 20$

Fig. 8. $\mathbb{H}$ w.r.t node density λ_{UE}

as few components as possible for longer times compared to the related schemes. It is worth noting that RRT−ORB and DJK−RRB achieve good performance in terms of network lifetime. The obtained results corroborate the previous ones depicted in Fig. 5 and Fig. 6. Indeed, the aforementioned strategies handle less traffic over the D2D sub-network due to their high service interruption level and packet loss rate. Consequently, nodes lifetime will be longer.

Fig. 8 illustrates the performance in terms of $\mathbb{H}$ metric which reflects the number of hops in the routing trees. This metric gives indication on the QoS presented to flows in terms of latencies where shorter is better. Specifically, the end-to-end and the average packet delays are in proportion to the product $\mathbb{H} \times T_{\text{SL}}$. Fig. 8a and Fig. 8b point out that the average number of hops increases almost linearly in accordance with the density of nodes λ_{UE}. As expected, DJK−RRB always yields the shortest number of hops by virtue of its strategy. We note that, in general, DJK−RRB and RRT−ORB lead to shorter paths and lower latencies compared with JRRA−EE. We recall, as illustrated above, that both DJK−RRB and RRT−ORB deteriorate service interruption and packet error rate metrics. In return, our proposal JRRA−EE outperforms its energy non-aware variant JRRA−EN.

In summary, network simulations show that JRRA−EE outperforms the variants in terms of network lifetime, packet loss and service interruption at the expense of small performance gaps with respect to latency and offloading capacity. Furthermore, JRRA−EE always outperforms its energy non-aware variant JRRA−EN.

VI. Conclusion

In this paper, we studied the LTE-D2D-based multihop offloading scheme for the intra-cell UE-to-UEs multicast/unicast flows. We proposed an energy-efficient offloading scheme to jointly solve the problem of multicast/unicast routing and the OFDMA resource block allocation. To increase the utility of the offloading system, the proposed scheme took into account the battery-limitation by defining an energy-budget for each cooperating UE. We also considered LTE-D2D-specific constraints: half-duplex operation and the contiguous resource block allocations. We formulated the problem as an ILP and we proposed a novel heuristic, named JRRA−EE, composed of a two-stage algorithm to solve it. We validated our proposal using the NS-3 simulator after implementing the whole LTE-D2D protocol stack. Through extensive simulations, we have shown that our proposed strategy JRRA−EE outperformed the related strategies. Performance gains manifested themselves as i) increased lifetime of the offloading network, ii) low packet loss and iii) low service interruption rates at the expense of small performance gaps in latency and offloading capacity.

References

[1] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception (Release 14)," 3rd Generation Partnership Project (3GPP), TS 36.101, Sep. 2017. [Online]. Available: http://www.3gpp.org/DynaReport/36101.htm

[2] G. Nemhauser and L. Wolsey, "Computational complexity," in *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988, pp. 114–145. [Online]. Available: http://dx.doi.org/10.1002/9781118627372.ch5

[3] T. Ta, J. S. Baras, and C. Zhu, "Improving smartphone battery life utilizing device-to-device cooperative relays underlaying lte networks," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 5263–5268.

[4] B. Liu, Y. Cao, W. Wang, and T. Jiang, "Energy budget aware device-to-device cooperation for mobile videos," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–7.

[5] A. Laha, X. Cao, W. Shen, X. Tian, and Y. Cheng, "An energy efficient routing protocol for device-to-device based multihop smartphone networks," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5448–5453.

[6] Z. Jingyi, L. Xi, and X. Quansheng, "Multi-hop routing for energy-efficiency enhancement in relay-assisted device-to-device communication," *The Journal of China Universities of Posts and Telecommunications*, vol. 22, no. 2, pp. 1–51, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S100588851560632X

[7] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (Release 14)," 3rd Generation Partnership Project (3GPP), TS 36.213, Sep. 2017. [Online]. Available: http://www.3gpp.org/DynaReport/36213.htm

[8] M. Hoeyhtyae, A. Maemmelae, U. Celentano, and J. Roening, "Power-efficiency in social-aware d2d communications," in *European Wireless 2016; 22th European Wireless Conference*, May 2016, pp. 1–6.

[9] M. Lauridsen, L. Noël, T. B. Sørensen, and P. Mogensen, "An empirical lte smartphone power model with a view to energy efficiency evolution," *Intel Technology Journal*, vol. 18, no. 1, pp. 172–193, 2014.

[10] N. Baldo, M. Miozzo, M. Requena-Esteso, and J. Nin-Guerrero, "An open source product-oriented LTE network simulator based on ns-3," in *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 2011, pp. 293–298.

[11] Y. d. J. Bultitude and T. Rautiainen, "IST-4-027756 WINNER II D1. 1.2 V1. 2 WINNER II Channel Models," 2007.

Dynamic load balancing in 5G HetNets for optimal performance-energy tradeoff

Misikir Eyob Gebrehiwot, Pasi Lassila and Samuli Aalto
Department of Communications and Networking
Aalto University, Finland
Email: {misikir.gebrehiwot, pasi.lassila, samuli.aalto}@aalto.fi

***Abstract*—We consider optimal energy-aware load balancing of elastic downlink data traffic inside a macrocell with multiple small cells within its coverage area. The model for the problem corresponds to a system of parallel M/M/1-PS queues, where the macrocell is represented by a multiclass M/M/1-PS queue and each small cell is an energy-aware M/M/1-PS queue with additional states for the idle timer and the so-called setup delay. We apply the theory of MDPs to develop a near-optimal state-dependent policy, both for a weighted sum of the performance and energy as well as for the constrained formulation, where energy is minimized subject to a constraint on the performance. Specifically, we utilize the first step of the well-known policy iteration method under which the routing decision for each arrival requires evaluating the marginal future cost of adding the arrival in the small cell or the macrocell. As our main contribution, we derive the associated value functions and the explicit form of the near-optimal FPI policy. The performance of the policy is illustrated through numerical examples.**

I. Introduction

Heterogeneous networks (HetNets) is one of the key enabling technologies for realizing the future 5G networks. Specifically, HetNets address the problem of spatially heterogeneous distribution of the traffic load within a cell by introducing inside the coverage region of the macrocell so-called small cells, sometimes also referred to as pico- or femtocells, with low-power base stations that are operating under the control of the macrocell. The small cells can offer at a traffic hot spot a high transmission rate to nearby users and thus some of the traffic can be off loaded away from the macrocell to the respective small cell. Such load balancing clearly can benefit the performance of the users, for example, by minimizing the delay. However, in modern systems it also important to consider the energy consumption of the system.

Such energy-aware load balancing in HetNets has been studied considerably recently. Typically, the approaches consider a fixed set of users or the optimization only takes into account average traffic parameters, see, e.g., [1], [2], [3], [4], [5]. However, such approaches do not take into account the randomly varying user population and the delay performance of the users. Our focus is to take these aspects explicitly into account by using a queueing theoretic approach.

In more detail, we study the following scenario. We consider a single macrocell with many small cells inside its coverage region serving downlink traffic. The user traffic consists of elastic flows, roughly corresponding to file transfers controlled by TCP, that are downloaded through the base stations. New flows arrive according to a Poisson process. Thus, the number of active flows varies randomly over time and the flow-level performance is represented by the mean flow-level delay, i.e., the mean file transfer delay. Upon the arrival of a new flow, a load balancing policy will decide whether to serve the flow through the local small cell or the macrocell. From the energy point of view, the macrocell is always on in order to provide coverage in the whole cell area. However, the small cells can be switched off to save energy during low load. Activating again a sleeping small cell incurs a performance cost in the form of a setup delay, thus giving rise to a performance-energy trade-off in the system. Our objective is to develop energy- and delay-aware load balancing algorithms.

In our flow-level model, the macrocell is represented by a multiclass M/M/1-PS queue, where the classes represent flows that arrived in a given small cell but are served by the macrocell. The small cells are modelled as a single-class M/M/1-PS queues with a setup delay, and we additionally allow another control parameter, the so-called idle timer, which defines how long a small cell waits before it falls into sleep mode after the small cell becomes idle (i.e., becomes empty of flows). Thus, the system consists of a set of parallel queues and the load balancing policy decides whether the arrival is routed to the small cell or the macrocell.

To characterize the performance-energy trade-off, we represent the system cost as a weighted sum of the delay and energy. To optimize the cost, we consider dynamic state-dependent policies and apply the theory of Markov Decision Processes (MDP). MDPs have been applied for the load balancing problem in HetNets in [6], [7], [8], [9], but they do not contain any explicit forms of the resulting policies, i.e., value iteration method is used to numerically solve the problem. Also, the system models are different to ours in these papers, or energy aspects are not considered. Note that in our earlier paper [10], we have applied the so-called FPI approach, to be discussed below, but the policy is obtained numerically and, in addition, the system model is slightly different (no idle timer in small cells and macrocell energy model is simpler).

As our main analytical contribution, we are able to define explicitly the (near) optimal policy. Specifically, we utilize the first step of the policy iteration algorithm [11], which gives the so-called FPI (First Policy Iteration) policy. By using a

ISBN 978-3-903176-08-9

static probabilistic policy as an initial policy, the cost under the FPI policy becomes separable and the routing decision depends only on the state of the local small cell and the state of the macrocell. The FPI policy is characterized by the so-called value functions of the macrocell and the small cells. We derive the explicit form of the value functions of the delay and power for the small cells represented by an M/M/1-PS queue with setup delay and idle timer. This extends our earlier analytical results for the M/M/1 queue with setup delay but no idle timer in [12]. Also, for the multiclass M/M/1-PS queue, we derive the value function of the power part, while the general form of the performance part is known, see [13], [14]. The resulting policy is still scalable as it only involves comparing the additional cost of allocating the flow in the macrocell or the small cell. The policy is near optimal since typically in policy iteration the largest gain over the initial policy is already obtained with the first iteration step.

Due to the explicit form of the value functions, insights can also be obtained from how the (near) optimal policy behaves. The marginal cost for the performance part in small cells has a weighted JSQ-like (Join-the-Shortest-Queue) structure with an additional constant factor that reflects the state of the server (setup or busy), while the marginal energy cost is just a constant independent of the setup/busy. In the macro cell, the marginal performance cost also has a similar linear form (JSQ-like) and the marginal energy cost is a constant.

In addition to formulating the optimization problem as a minimization problem of the weighted sum, we formulate the problem as a constrained optimization problem. We apply the theory of constrained MDPs [15], [16] and consider minimizing the mean power subject to a constraint on the mean delay. This formulation is often more natural than the weighted sum variant, where the weight is arbitrary. We observe that, by Lagrangian relaxation of the constraint, the constrained problem can be interpreted as the minimization of the weighted sum of the power and delay, where the weight must be determined to satisfy the constraint. Thus, we can apply our results for the FPI policy and we give an iterative algorithm to obtain the optimal weight parameter that defines the (near) optimal policy and solves the constrained problem.

We highlight the properties of the dynamic policies through numerical examples. In particular, we are interested in the impact of the idle timer, which has not been studied previously in this context. We have shown already in our earlier paper that with a single energy-aware M/G/1-PS queue the optimal value of the idle timer is always either zero or infinity for the weighted sum cost function, see [17]. Our numerical results indicate that the same remains true also in this much more complex scenario with parallel queues and non-Poisson arrivals. However, in the case of the constrained optimization formulation, if the setup delay is short enough, a strictly positive and finite optimal choice for the idle timer exists.

The paper is organized as follows. The system model is given in Section II. The optimal load balancing problem is defined in Section III, where also our main analytical contributions are derived. Section IV considers the load balancing problem as a constrained MDP. Numerical results and conclusions are in Sections V and VI, respectively.

II. Model

We consider a heterogeneous wireless system consisting of a single macrocell and K separate small cells located inside the coverage area of the macrocell. The macrocell is indexed by 0 and the small cells by $k = 1, \ldots, K$. We assume that the small cells operate in an outband mode, i.e., they have their own radio resources and do not interfere with the macrocell. In addition, we assume that the small cells are far enough from each other so that they do not interfere with each other either.

Traffic consists of elastic downlink data flows (such as TCP file transfers). Let λ_k denote the arrival rate of new flows within the area of small cell k. Each such a flow can be served either by the small cell itself or the macrocell (but not by the other small cells). Upon the arrival, a routing decision must be made whether the flow is attached to the small cell or the macrocell. In addition, let λ_0 denote the arrival rate of those flows (outside the "hotspot" areas covered by the small cells) that can only be served by the macrocell. All the arrival processes are assumed to be independent Poisson processes.

Each small cell k is modeled by a single server PS queue, which implies that flows are scheduled in each time slot so that all resources are given to one flow at a time and the flows are served in a round-robin manner. We assume that the service time S_k of an arbitrary flow in small cell k is exponentially distributed with mean $E[S_k] = 1/\mu_k$. The mean service time $E[S_k]$ is the average time needed to complete the transfer of a random flow if there were no other flows to be carried by the same cell. We note that the service time is affected at least by the size of the original flow, the location of the corresponding mobile terminal (within the small cell), and the radio channel conditions during the flow transfer. However, since the scheduler of the small cell does not utilize these features, we do not model them separately.

For each small cell, we apply the DELAYEDOFF sleep state control (according to the terminology of [18]). As long as there are flows in the small cell to be served, the state of the server is said to be BUSY, but as soon as the system becomes empty, the state changes to IDLE. The server remains IDLE as long as one of the following events take place. Either a new flow arrives, in which case the server becomes again BUSY and starts serving the new flow, or the idle timer (associated with the idle server) expires, in which case the server is immediately switched OFF. In our model, the length I_k of the idle timer of small cell k is assumed to be independently and exponentially distributed with mean $E[I_k] = 1/\omega_k$. In the latter case, the server remains OFF until a new flow is routed to small cell k and the server is put to the SETUP state. After a setup delay D_k, which in our model is assumed to be independently and exponentially distributed with mean $E[D_k] = 1/\delta_k$, the server becomes again BUSY and starts serving the waiting flows.

The state of small cell k at time t is described by the pair $(X_k(t), B_k(t))$, where $X_k(t)$ refers to the number of flows

and $B_k(t)$ to the state of the server. Note that server k is

$$\begin{array}{ll} \text{BUSY}, & \text{if } X_k(t) > 0 \text{ and } B_k(t) = 1; \\ \text{IDLE}, & \text{if } X_k(t) = 0 \text{ and } B_k(t) = 1; \\ \text{OFF}, & \text{if } X_k(t) = 0 \text{ and } B_k(t) = 0; \\ \text{SETUP}, & \text{if } X_k(t) > 0 \text{ and } B_k(t) = 0. \end{array}$$

We denote the power consumption in these energy states of small cell k by $P_k^{\mathrm{E-STATE}}$, and we assume that

$$0 = P_k^{\mathrm{o}} < P_k^{\mathrm{i}} \leq \min\{P_k^{\mathrm{s}}, P_k^{\mathrm{b}}\},$$

where superscripts o, i, s, and b refer to OFF, IDLE, SETUP, and BUSY states, respectively.

The macrocell (0) serves $K+1$ different classes of flows. Class 0 refers to those flows that can only be served by the macrocell and class k to those flows that arrive in the area of small cell k but are routed to the macrocell. The macrocell itself is modeled by a single server multiclass PS queue, which again implies that flows are scheduled in each time slot so that all resources are given to one flow at a time and the flows are served in a round-robin manner. We assume that the service time $S_{0,k}$ of an arbitrary flow in class $k \in \{0, 1, \ldots, K\}$ is exponentially distributed with mean $1/\mu_{0,k}$.

For the macrocell, we do not apply any sleep state control, but it is a NEVEROFF server (according to the terminology of [18]). Thus, the state of the macrocell at time t is described by the vector $X_0(t) = (X_{0,0}(t), X_{0,1}(t), \ldots, X_{0,K}(t))$, where $X_{0,k}(t)$ refers to the number of flows in class k. Note that server 0 is

$$\begin{array}{ll} \text{BUSY}, & \text{if } |X_0(t)| > 0; \\ \text{IDLE}, & \text{if } |X_0(t)| = 0, \end{array}$$

where

$$|X_0(t)| = X_{0,0}(t) + X_{0,1}(t) + \ldots + X_{0,K}(t)$$

denotes the total number of flows in macrocell. The power consumption of macrocell is denoted by $P_0^{\mathrm{E-STATE}}$, and we assume that

$$0 < P_0^{\mathrm{i}} \leq P_0^{\mathrm{b}}.$$

III. Optimal load balancing problem

Our objective is to develop dynamic, state-dependent load balancing policies that simultaneously take into account both the power consumption and the delay of the flows. The load balancing policy decides for each arriving flow in the small cells, whether the arrival is served by the local small cell or the macrocell. For our model, optimal policies can be developed in the framework of MDPs [11].

Before discussing the optimization, we state a necessary condition for the stability of the system. As in [10], the macrocell is the bottleneck in the system, because it serves as an overflow system for the arrivals in the small cells, and thus, the maximal stability condition for any load balancing policy is given by

$$\frac{\lambda_0}{\mu_{0,0}} + \sum_{k=1}^{K} \frac{(\lambda_k - \mu_k)^+}{\mu_{0,k}} < 1. \tag{1}$$

For the optimization, the cost rates with respect to the performance (delay) and the energy need to be defined first. Let the vector $x_0 = (x_{0,0}, \ldots, x_{0,K})$ denote a given state of the macrocell. Similarly, we denote by $x = (x_1, \ldots, x_K)$ and $b = (b_1, \ldots, b_K)$ vectors for the number of flows and the state of the server in each small cell. Given that there are (x_o, x) flows in the system, the instantaneous cost rate for the performance, $c^{\mathrm{p}}(x_0, x)$, is given by

$$c^{\mathrm{p}}(x_0, x) = \sum_{k=0}^{K} x_{0,k} + \sum_{k=1}^{K} x_k, \tag{2}$$

i.e., it is the total number of flows in the system. For the energy, the instantaneous cost rate, $c^{\mathrm{e}}(x_0, x, b)$, is the total instantaneous power in the given state and it equals

$$\begin{aligned} c^{\mathrm{e}}(x_0, x, b) = {} & 1_{|x_0|=0} P_0^{\mathrm{i}} + 1_{|x_0|>0} P_0^{\mathrm{b}} + \\ & \sum_{k=1}^{K} (1_{x_k>0, b_k=1} P_k^{\mathrm{b}} + 1_{x_k=0, b_k=1} P_k^{\mathrm{i}} + 1_{x_k>0, b_k=0} P_k^{\mathrm{s}}), \end{aligned} \tag{3}$$

where 1_A denotes the indicator function of the event A. To characterize the trade-off between performance and energy, the total instantaneous cost rate in state (x_0, x, b), $c(x_0, x, b)$, is defined as the weighted sum of performance and energy,

$$c(x_0, x, b) = w_1 c^{\mathrm{p}}(x_0, x) + w_2 c^{\mathrm{e}}(x_0, x, b),$$

where $w_1, w_2 \geq 0$ are the weight parameters.

We denote by π the set of all possible load balancing policies that are stable under the condition (1). For a given policy π, let $E[X^\pi]$ and $E[P^\pi]$ denote the mean total number of flows in the system and the resulting mean power consumption, respectively. Our objective is to consider the following optimization problem,

$$\min_{\pi} w_1 E[X^\pi] + w_2 E[P^\pi]. \tag{4}$$

Note that by dividing (4) with the total arrival rate $\sum_k \lambda_k$, the objective is, by Little's law, equal to to the weighted sum of the mean flow delay and the mean energy per flow.

A. Optimal dynamic policy

The optimization problem (4) is an MDP and can be solved numerically iteratively with the policy iteration method [11], as follows. Let y denote a vector of the entire state of the system, i.e., $y = (x_0, x, b)$. In the load balancing policy, associated with each state (x_0, x, b) there is a set of actions $\mathcal{A}(y)$ relating to the decisions where the arrivals are routed. Let π_n denote the policy at the n^{th} iteration step. The iterated policy policy at the next step, π_{n+1}, is obtained by solving in each state y the following optimality equation

$$\begin{aligned} & \pi_{n+1}(y) = \\ & \arg\min_{a \in \mathcal{A}(y)} \left(c(y) - \bar{c}^{\pi_n} + \sum_{y'} q_{y,y'}(a) v^{\pi_n}(y') \right), \forall y, \end{aligned} \tag{5}$$

where $\bar{c}^{\pi_n}$ is the mean cost under policy π_n, $q_{y,y'}(a)$ is the transition rate from state y to state y' when action a is taken

and $v^{\pi_n}(y')$ is the so-called value function of state y' for policy π_n.

The value function of each state under given policy π gives the mean difference in the cost when starting initially the process from the state $\mathbf{y}$ and the long term average cost $\bar{c}^{\pi}$. The value function of each state are, on the other hand, characterized by the following set of linear equations

$$c(y) - \bar{c}^{\pi} + \sum_{y'} q_{y,y'}(a^{\pi}(y))(v^{\pi}(y') - v^{\pi}(y)) = 0, \ \forall y, \quad (6)$$

where $a^{\pi}(y)$ is the action taken in state y under policy π.

The policy iteration algorithm can be started from any stable initial policy π^0, for which the value functions are first solved from (6). Then the iterated policy π^1 is obtained from (5), and the process repeats. The policy iteration is guaranteed to converge to the optimal policy in a finite number of iterations [11]. Unfortunately, for the state space consists of $(3K+1)$ dimensions which makes it impossible to solve the optimal policy numerically.

B. Near-optimal FPI policy

In the FPI (First Policy Iteration) approach, the idea is that by selecting the initial policy appropriately the first step of the optimization (5) can be carried out explicitly. Typically this first step already gives the largest improvement, which makes the FPI policy near optimal.

Consider now a probabilistic policy determined by the vector $p = (p_1, \ldots, p_K)$, where each component p_k gives the probability to route the incoming class-k flow to the small cell k and with probability $(1 - p_k)$ the flow is routed to the macrocell. By using a probabilistic policy as the initial policy renders the stochastic behavior of the macrocell and the small cells independent of each other. Thus, the relative value of the state $y = (x_0, x, b)$ can be expressed as

$$v(x_0, x, b) = v_0(x_0) + \sum_{k=1}^{K} v_k(x_k, b_k), \quad (7)$$

where $v_0(x_0)$ and $v_k(x_k, b_k)$ are the value functions of the macrocell and small cell k, respectively.

A reasonable selection for the initial probabilistic policy is to balance the load in all the cells, as much as possible. We denote this policy by p^{LB}. Assume that the classes of the small cells, $k = 1, \ldots, K$, are ordered in a descending order according to the cell loads, i.e., $\lambda_1/\mu_1 > \cdots > \lambda_K/\mu_K$. Note that for all those small cells where the load is already less than the load of the macrocell when serving its own traffic, no traffic can be moved to the macrocell, i.e., the corresponding $p_k^{\mathrm{LB}} = 1$. Thus, let k^* denote the index value of the last small cell from which traffic can be moved to the macrocell, i.e.,

$$k^* = \{\max k = 1, \ldots, K : \lambda_k/\mu_k > \lambda_0/\mu_{0,0}\}.$$

It is easy to see that the load is then equalized by setting

$$\begin{cases} p_k^{\mathrm{LB}} = \frac{\mu_k}{\lambda_k} \cdot \frac{\frac{\lambda_0}{\mu_{0,0}} + \sum_{k=1}^{k^*} \frac{\lambda_k}{\mu_{0,k}}}{1 + \sum_{k=1}^{k^*} \frac{\mu_k}{\mu_{0,k}}}, & k = 1, \ldots, k^*, \\ p_k^{\mathrm{LB}} = 1, & k = k^*+1, \ldots, K. \end{cases}$$

The optimization (5) only concerns the arrival events. Thus, the decision to serve the arrival in small cell k or route it to the macrocell simply consists of evaluating the additional cost of adding the arrival to the small cell or to the macrocell. Due to the separable form of the value function (7), the action to serve the arrival in the macrocell or the small cell k in state (x_0, x, b) is given by

$$a_k^{\mathrm{FPI}}(x_0, x, b) = \quad (8)$$
$$\begin{cases} \text{macrocell}, & \text{if } v_0(x_{0,0}, \ldots, x_{0,k}+1, \ldots, x_{0,K}) - \\ & \quad v_0(x_{0,0}, \ldots, x_{0,K}) < \\ & \quad v_k(x_k+1, b_k) - v_k(x_k, b_k) \\ \text{small cell}, & \text{otherwise.} \end{cases}$$

In summary, the main advantage in the FPI approach is that it allows us to systematically construct an optimized dynamic policy, which is near optimal and (almost) fully explicit, as will be seen in the following section. Note that through our results, the FPI policy is also fully specified in the entire state space of the system, i.e., there is no need for any truncation to evaluate the mean cost under the FPI policy by simulation.

C. Value functions

Here we present the main analytical results of the paper: explicit value functions for the performance and energy for the M/M/1-PS DELAYDOFF and multiclass M/M/1-PS models.

1) M/M/1-PS DELAYEDOFF: Consider a generic M/M/1-PS DELAYEDOFF queue with arrival rate λ, mean service time $E[S] = 1/\mu$, mean idle timer $E[I] = 1/\omega$, and mean setup delay $E[D] = 1/\delta$. Assume that the system is stable, i.e., $\rho < 1$, where load $\rho = \lambda E[S]$. In addition, let P^{o}, P^{i}, P^{s}, and P^{b} denote the power consumption in energy states OFF, IDLE, SETUP, and BUSY, respectively.

Proposition 1: For a stable M/M/1-PS DELAYEDOFF queue, the relative value function with respect to performance in state (n, b) is given by

$$\begin{aligned} v^{\mathrm{p}}(n,1) - v^{\mathrm{p}}(0,0) &= \frac{E[S]n(n+1)}{2(1-\rho)} \\ &- \frac{E[D]n\rho(1+\lambda E[D])}{(1-\rho)(1+\lambda E[D]+\lambda E[I])} \\ &- \frac{E[I]\lambda E[D]}{(1-\rho)(1+\lambda E[D]+\lambda E[I])}; \\ v^{\mathrm{p}}(n,0) - v^{\mathrm{p}}(0,0) &= \frac{E[S]n(n+1)}{2(1-\rho)} \\ &+ \frac{E[D]n(1+\lambda E[D])}{1+\lambda E[D]+\lambda E[I]} \\ &+ \frac{E[I](n-1)\lambda E[D]}{(1-\rho)(1+\lambda E[D]+\lambda E[I])}. \end{aligned}$$

Proof: The Howard equations for the system are:

$$\begin{aligned} -\bar{c}^{\mathrm{p}} + \lambda(v^{\mathrm{p}}(1,0) - v^{\mathrm{p}}(0,0)) &= 0, \\ n - \bar{c}^{\mathrm{p}} + \lambda(v^{\mathrm{p}}(n+1,0) - v^{\mathrm{p}}(n,0)) + & \\ \delta(v^{\mathrm{p}}(n,1) - v^{\mathrm{p}}(n,0)) &= 0, \quad n \geq 1, \end{aligned}$$

$$
\begin{aligned}
&n - \bar{c}^{\mathrm{p}} + \lambda(v^{\mathrm{p}}(n+1,1) - v^{\mathrm{p}}(n,1)) + \\
&\quad \mu(v^{\mathrm{p}}(n-1,1) - v^{\mathrm{p}}(n,1)) = 0, \quad n \geq 1, \\
&-\bar{c}^{\mathrm{p}} + \lambda(v^{\mathrm{p}}(1,1) - v^{\mathrm{p}}(0,1)) + \\
&\quad \omega(v^{\mathrm{p}}(0,0) - v^{\mathrm{p}}(0,1)) = 0,
\end{aligned}
$$

where $\bar{c}^{\mathrm{p}}$ denotes the mean number of flows given by

$$
\bar{c}^{\mathrm{p}} = E[X] = \lambda E[T] = \frac{\rho}{1-\rho} + \frac{\lambda E[D](1+\lambda E[D])}{1+\lambda E[D]+\lambda E[I]}.
$$

Now it is a straightforward task to verify that these equations are satisfied by the proposed relative value function. ■

Corollary 1: For a stable M/M/1-PS DELAYEDOFF queue, the marginal performance cost in state (n,b) is given by

$$
\begin{aligned}
v^{\mathrm{p}}(n+1,1) - v^{\mathrm{p}}(n,1) &= \frac{E[S](n+1)}{1-\rho} \\
&- \frac{E[D]\rho(1+\lambda E[D])}{(1-\rho)(1+\lambda E[D]+\lambda E[I])}, \quad n \geq 0; \\
v^{\mathrm{p}}(1,0) - v^{\mathrm{p}}(0,0) &= \frac{E[S]}{1-\rho} \\
&+ \frac{E[D](1+\lambda E[D])}{1+\lambda E[D]+\lambda E[I]}; \\
v^{\mathrm{p}}(n+1,0) - v^{\mathrm{p}}(n,0) &= \frac{E[S](n+1)}{1-\rho} \\
&+ \frac{E[D](1+\lambda E[D])}{1+\lambda E[D]+\lambda E[I]} \\
&+ \frac{E[I]\lambda E[D]}{(1-\rho)(1+\lambda E[D]+\lambda E[I])}, \quad n \geq 1.
\end{aligned}
$$

Proposition 2: For a stable M/M/1-PS DELAYEDOFF queue, the relative value function with respect to the energy in state (n,b) is given by

$$
\begin{aligned}
v^{\mathrm{e}}(n,1) - v^{\mathrm{e}}(0,0) &= n \cdot \gamma \\
&+ \frac{E[I]((P^{\mathrm{i}} - P^{\mathrm{o}}) + \lambda E[D](P^{\mathrm{i}} - P^{\mathrm{s}}))}{1+\lambda E[D]+\lambda E[I]}, \\
v^{\mathrm{e}}(n,0) - v^{\mathrm{e}}(0,0) &= n \cdot \gamma \\
&+ \frac{E[D]((P^{\mathrm{s}} - P^{\mathrm{o}}) + E[I](P^{\mathrm{i}} - P^{\mathrm{o}}))}{1+\lambda E[D]+\lambda E[I]}, \\
\gamma &= \frac{E[S]((P^{\mathrm{b}} - P^{\mathrm{o}}) + \lambda E[D](P^{\mathrm{b}} - P^{\mathrm{s}}))}{1+\lambda E[D]+\lambda E[I]} \\
&+ \frac{E[S]\lambda E[I](P^{\mathrm{b}} - P^{\mathrm{i}})}{1+\lambda E[D]+\lambda E[I]}.
\end{aligned}
$$

Proof: The Howard equations for the system are:

$$
\begin{aligned}
&P^{\mathrm{o}} - \bar{c}^{\mathrm{e}} + \lambda(v^{\mathrm{e}}(1,0) - v^{\mathrm{e}}(0,0)) = 0, \\
&P^{\mathrm{s}} - \bar{c}^{\mathrm{e}} + \lambda(v^{\mathrm{e}}(n+1,0) - v^{\mathrm{e}}(n,0)) + \\
&\quad \delta(v^{\mathrm{e}}(n,1) - v^{\mathrm{e}}(n,0)) = 0, \quad n \geq 1, \\
&P^{\mathrm{b}} - \bar{c}^{\mathrm{e}} + \lambda(v^{\mathrm{e}}(n+1,1) - v^{\mathrm{e}}(n,1)) + \\
&\quad \mu(v^{\mathrm{e}}(n-1,1) - v^{\mathrm{e}}(n,1)) = 0, \quad n \geq 1, \\
&P^{\mathrm{i}} - \bar{c}^{\mathrm{e}} + \lambda(v^{\mathrm{e}}(1,1) - v^{\mathrm{e}}(0,1)) + \\
&\quad \omega(v^{\mathrm{e}}(0,0) - v^{\mathrm{e}}(0,1)) = 0,
\end{aligned}
$$

where $\bar{c}^{\mathrm{e}}$ denotes the mean power consumption given by

$$
\bar{c}^{\mathrm{e}} = E[P] = \rho P^{\mathrm{b}} + (1-\rho)\frac{P^{\mathrm{o}} + \lambda E[D]P^{\mathrm{s}} + \lambda E[I]P^{\mathrm{i}}}{1+\lambda E[D]+\lambda E[I]}.
$$

It is again a straightforward task to verify that these equations are satisfied by the proposed relative value function. ■

Corollary 2: For a stable M/M/1-PS DELAYEDOFF queue, the marginal energy cost in state (n,b) is given by

$$
\begin{aligned}
&v^{\mathrm{e}}(n+1,1) - v^{\mathrm{e}}(n,1) = \gamma, \quad n \geq 0; \\
&v^{\mathrm{e}}(1,0) - v^{\mathrm{e}}(0,0) = \gamma \\
&\quad + \frac{E[D]((P^{\mathrm{s}} - P^{\mathrm{o}}) + E[I]((P^{\mathrm{i}} - P^{\mathrm{o}})}{1+\lambda E[D]+\lambda E[I]}, \\
&v^{\mathrm{e}}(n+1,0) - v^{\mathrm{e}}(n,0) = \gamma, \quad n \geq 1,
\end{aligned}
$$

where γ is defined in Proposition 2.

The explicit value functions for the M/M/1-PS DELAYEDOFF model are novel results and not available in the literature. The results in Corollary 1 and 2 are used when evaluating the marginal cost of serving the flow in small cell k, in (8). In the marginal cost expressions of Corollary 1 and 2, the arrival rate in small cell k after the initial policy is $\lambda_k = p_k^{\mathrm{LB}}\lambda$.

Also, observe that in Corollary 1, the marginal cost with respect ot the performance has a linear cost with respect to the number of jobs, i.e., similarly to the JSQ rule (Join-the-Shortest-Queue), but in addition there is a positive or negative constant factor depending on whether the server is in sleep/setup state or busy state. Thus, it is from the future cost better to keep an already busy server busy than to wake it up, which is also logical. On the other hand, by Corollary 2, the marginal energy cost is interestingly constant and does not depend on busy/setup state, unless the server is switched off.

2) MULTICLASS M/M/1-PS NEVEROFF: Consider a generic multiclass M/M/1-PS NEVEROFF queue with $K+1$ classes of customers indexed by $k = 0, 1, \ldots, K$. Let λ_k and $E[S_k] = 1/\mu_k$ denote the arrival rate and the mean service time for class k, respectively. In addition, let $\lambda = \lambda_0 + \lambda_1 + \ldots + \lambda_K$ denote the total arrival rate and

$$
E[S] = \frac{1}{\lambda}(\lambda_0 E[S_0] + \lambda_1 E[S_1] + \ldots + \lambda_K E[S_K])
$$

refer to the mean service time over all the customers. Assume that the system is stable, i.e., $\rho < 1$, where load $\rho = \lambda E[S]$. Finally, let P^{i} and P^{b} denote the power consumption in energy states IDLE and BUSY, respectively.

Proposition 3: For a stable multiclass M/M/1-PS NEVEROFF queue, the relative value function with respect to performance in state $\mathbf{n} = (n_0, n_1, \ldots, n_K)$ is given by

$$
v^{\mathrm{p}}(\mathbf{n}) - v^{\mathrm{p}}(\mathbf{0}) = \sum_{k=0}^{K} a_k(n_k^2 + n_k) + \sum_{k=0}^{K-1}\sum_{\ell=k+1}^{K} 2a_{k,\ell}n_k n_\ell,
$$

where the coefficients a_k and $a_{k,\ell}$ ($k < \ell$) are solved from the following system of linear equations:

$$1 + 2\sum_{i=0}^{K} \lambda_i a_{k,i} - 2\mu_k a_k = 0, \quad 0 \le k \le K;$$
$$1 + \sum_{i=0}^{K} \lambda_i (a_{k,i} + a_{\ell,i}) - (\mu_k + \mu_\ell) a_{k,\ell} = 0,$$
$$0 \le k < \ell \le K,$$

with notations $a_{k,k} = a_k$ and $a_{k,\ell} = a_{\ell,k}$ for any k, ℓ.

Proof: The general result for a multiclass M/M/1-PS NEVEROFF queue was proved in [13]. ∎

Proposition 4: For a stable multiclass M/M/1-PS NEVEROFF queue, the relative value function with respect to energy in state $\mathbf{n} = (n_0, n_1, \ldots, n_K)$ is given by

$$v^{\mathrm{e}}(\mathbf{n}) - v^{\mathrm{e}}(\mathbf{0}) = \sum_{k=0}^{K} E[S_k] n_k (P^{\mathrm{b}} - P^{\mathrm{i}}),$$

where $\mathbf{0} = (0, 0, \ldots, 0)$ is the null vector.

Proof: The Howard equations for the system read as follows:

$$P^{\mathrm{i}} - \bar{c}^{\mathrm{e}} + \sum_{k=0}^{K} \lambda_k (v^{\mathrm{e}}(\mathbf{e}_k) - v^{\mathrm{e}}(\mathbf{0})) = 0,$$
$$P^{\mathrm{b}} - \bar{c}^{\mathrm{e}} + \sum_{k=0}^{K} \frac{n_k \mu_k}{n_0 + \ldots + n_K} (v^{\mathrm{e}}(\mathbf{n} - \mathbf{e}_k) - v^{\mathrm{e}}(\mathbf{n})) +$$
$$\sum_{k=0}^{K} \lambda_k (v^{\mathrm{e}}(\mathbf{n} + \mathbf{e}_k) - v^{\mathrm{e}}(\mathbf{n})) = 0, \quad \mathbf{n} \neq \mathbf{0},$$

where $\mathbf{e}_k$ is a unit vector into direction k and $\bar{c}^{\mathrm{e}}$ denotes the mean power consumption given by

$$\bar{c}^{\mathrm{e}} = E[P] = \rho P^{\mathrm{b}} + (1 - \rho) P^{\mathrm{i}}.$$

It is again a straightforward task to verify that these equations are satisfied by the proposed relative value function. ∎

Corollary 3: For a stable multiclass M/M/1-PS NEVEROFF queue, the marginal cost with respect to energy in state $\mathbf{n} = (n_0, n_1, \ldots, n_K)$ is given by

$$v^{\mathrm{e}}(\mathbf{n} + \mathbf{e}_k) - v^{\mathrm{e}}(\mathbf{n}) = E[S_k](P^{\mathrm{b}} - P^{\mathrm{i}}),$$

where $\mathbf{e}_k$ is the unit vector into direction k.

The explicit form of the value function for the energy part in the multiclass M/M/1-PS model is a again novel result and not available in the literature. The results in Proposition 3 and Corollary 3 are used when evaluating the additional cost of serving the flow in the macrocell in (8). Note that after the initial policy each class $k = 1, \ldots, K$ has arrival rate $\lambda_k = (1 - p_k^{\mathrm{LB}})\lambda$.

As the value function for the performance has a quadratic form, see Proposition 3, it is clear that the marginal performance cost $v^{\mathrm{p}}(\mathbf{n} + \mathbf{e}_k) - v^{\mathrm{p}}(\mathbf{n})$ has in general a linear form. On the other hand, by Corollary 3, the marginal energy cost is constant.

IV. Constrained optimization problem

In Section III the performance-energy tradeoff was characterized by a weighted sum of the performance and the energy (4). In this formulation, defining appropriate weights in a given setting may be difficult. Instead, a more natural characterization of the tradeoff can be to consider the following constrained optimization formulation: the objective is to determine the policy π that minimizes the energy consumption subject to a constraint of the maximum mean delay $T^{\max}$,

$$\min_{\pi} E[P^{\pi}] \quad \text{s.t.} \ \frac{1}{\sum_k \lambda_k} E[X^{\pi}] \le T^{\max}, \tag{9}$$

where the instantaneous costs for the performance and energy are given by (2) and (3), respectively. Note that above again Little's result has been applied in the delay consraint.

The optimization problem (9) can be analyzed as a constrained MDP [15], [16] by applying standard Langrangian techniques. The idea is to introduce the so-called Lagrange parameter $\beta \ge 0$, and consider the following form for the immediate costs

$$c(x_0, x, b; \beta) = c^{\mathrm{p}}(x_0, x) + \beta c^{\mathrm{e}}(x_0, x, b).$$

In this way, for any given fixed value of β, the problem is in fact a standard unconstrained MDP with a weighted objective function (4) as defined in Section III, and the optimal policy $\pi^*(\beta)$ can be obtained by using, e.g., the policy iteration algorithm. Assuming that a feasible solution to (9) exists, there exists an optimal β^* such that the resulting policy $\pi^*(\beta^*)$ is also the optimal solution to (9), or it may be that to satisfy the constraint in (9) as an equality, the optimal policy is an appropriately randomized policy between two policies associated with β_{L}^* and β_{H}^*, such that $E[X^{\pi^*(\beta_{\mathrm{L}}^*)}]/\sum_k \lambda_k < T^{\max}$ and $E[X^{\pi^*(\beta_{\mathrm{H}}^*)}]/\sum_k \lambda_k > T^{\max}$, respectively, see [16], [15].

We have applied the following simple iterative algorithm to determine β^*, similarly as in [19]. Let n here denote the iteration round. Given the current value β_n at the n^{th} iteration, the next value β_{n+1} is obtained from

$$\beta_{n+1} = \beta_n + \frac{1}{n} \cdot \left(\frac{E[X^{\pi(\beta_n)}]}{\sum_k \lambda_k} - T^{\max} \right). \tag{10}$$

The iteration is stopped after a given number of iterations. Additionally, for each value of β_n, due to the high dimensionality of the state space in our problem, the policy iteration can not be iterated until the optimal policy is found. Instead, we apply only the FPI policy for which the explicit expressions have been given in Section III for the weighted cost. Also due to the enormous size of the state space, the performance and energy costs are obtained through simulation. This gives us finally the near optimal policy and the simulated solution to the original constrained optimization problem (9).

To illustrate the constrained optimization approach, we consider the simple model with one small cell with the following parameters: $E[I] = 1, E[D] = 0.1, \lambda_1 = 12.6, \mu_1 = 18.73$, $P_1^{\mathrm{b}} = P_1^{\mathrm{s}} = 100, P_1^{\mathrm{i}} = 70$ and $P_1^{\mathrm{o}} = 0$. In the macrocell, there

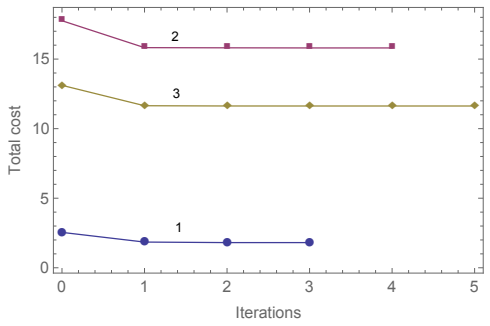


Fig. 1. Illustration of the convergence of the cost in the policy iteration as a function of the iterations (x-axis) and the convergence of the Lagrange parameter β for first three β-iterations (labeled as 1,2 and 3) in the constrained MDP.

are thus two classes with following parameters: $\lambda_0 = 2, \mu_{0,0} = 12.34, \mu_{0,1} = 6.37, P_0^{\mathrm{b}} = 1000$ and $P_0^{\mathrm{i}} = 700$. Figure 1 illustrates the policy iteration and the iteration with respect to β. In the figure, we show the result of the convergence of the policy iteration for the first three values of β. The initial value of β corresponds to the curve labeled 1 and on the x-axis the policy iteration convergence is shown as a function of the iteration count. In this case, the policy converged with 3 iterations. Then β is adapted according to (10) and the policy iteration is applied until convergence, see curve labeled 2, etc. In this small example, the policy iteration can be numerically evaluated until convergence. Later in our numerical results this is not possible due to the enormous state space and we apply the FPI policy only, which is then simulated to obtain the cost. However, observe that the largest improvement is always obtained with the first step of the policy iteration, i.e., the FPI policy, and the reduction in cost after that is marginal. Thus, we can argue that the FPI policy is indeed near optimal.

V. Numerical results

Here, we study the performance of proposed policies through simulation runs. We consider a system consisting of one macro and four small cells. Small cells can be in a sleep state where they do not consume power. Otherwise, power consumption of small cell k in the busy, setup and idle states are $P_k^{\mathrm{b}} = P_k^{\mathrm{s}} = 100$ and $P_k^{\mathrm{i}} = 70$ W, respectively. The macro cell consumes 1000 W when it is busy and 700 W when idle.

Service rate for a request originating from within a small cell is $\mu_k = 18.73$ s^{-1} if it is served by the small cell, and $\mu_{0,k} = 6.37$ s^{-1} if offloaded to the macro. Additionally, the macro cell also serves users that are outside the coverage area of the small cells with a rate $\mu_0 = 12.34$ s^{-1}, obtained by assuming file sizes of 5 Mb and typical measured mean channel qualities, see [10] for more detailed justifications. In the entire simulation study we set arrival rate in the macro cell to $\lambda_0 = 2$ s^{-1}, and systematically choose small cell arrival rates so that the load on the small cell is $0.05, 0.2, 0.32,$ and 0.5.

We study the impact of mean idle timer ($E[I]$) and mean setup delay ($E[D]$) on the performance of the proposed FPI policy. We start by considering unconstrained minimization of the weighed sum cost in the following section and then the constrained problem.

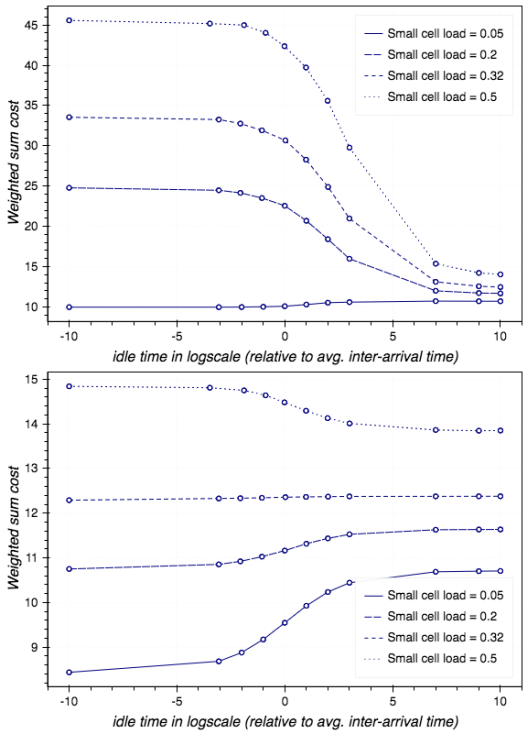


Fig. 2. Energy response time weighted sum cost for $E[D] = 1$ (upper figure) and $E[D] = 0.05$ (lower figure). Mean idle timer (x-axis) is shown in base-2 logarithmic scale.

A. Weighted sum cost

Figure 2 shows the weighted sum cost of mean response time and mean power consumption as a function of idle timer for two mean setup delay values, $E[D] = 1$ (upper figure) and $E[D] = 0.05$ (lower figure), and weight $\beta = 0.01$, i.e., $w_1 = 1$ and $w_2 = 0.01$. That is, the employed policy is $\pi^{\mathrm{FPI}}(0.01)$. The FPI policy is guaranteed to decrease the cost relative to the initial policy, which is the static load balancing policy. In this setting, the reduction in the cost is typically approximately 10%, e.g., with $E[D] = 1$ and $E[I] = 0$ at load $\rho = 0.5$ the corresponding static LB cost is 50.5, while the cost of the FPI policy is 46. However, the gain can be significantly affected by the choice of weight parameter β; decreasing β would increase the gain.

A more important aspect here is the impact of the mean idle timer. When setup delay is long, weighted sum cost decreases as a function of idle timer except at low load. Similarly in the short setup delay case, weighted sum cost remains monotonous as a function of idle timer, and whether it is an increasing or decreasing function depends on the load. Even though the unit on the x-axis in Figure 2 is the mean interarrival time (in log-2 base), these properties remain the same even if the curves would be shown as a function of the absolute value of the mean idle timer. Thus, in this scenario with FPI policy, the optimal configuration (with respect to idle timer) is either

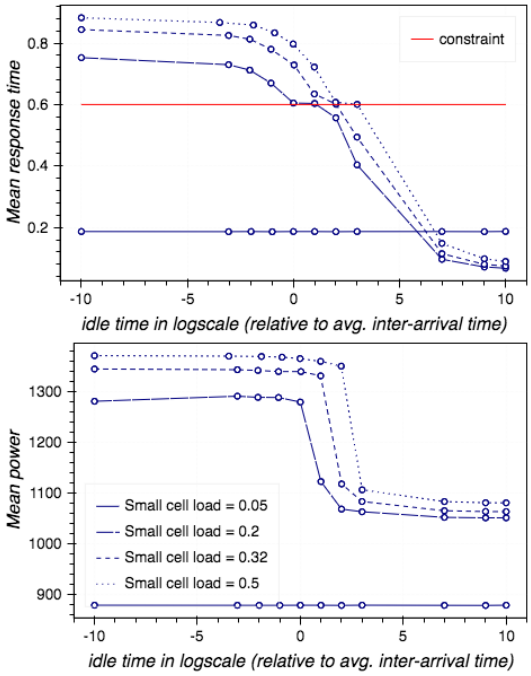


Fig. 3. Performance of FPI(β^*) policy as a function of mean idle timer with $E[D] = 1$ s. Mean idle timer (x-axis) is shown in base-2 logarithmic scale.

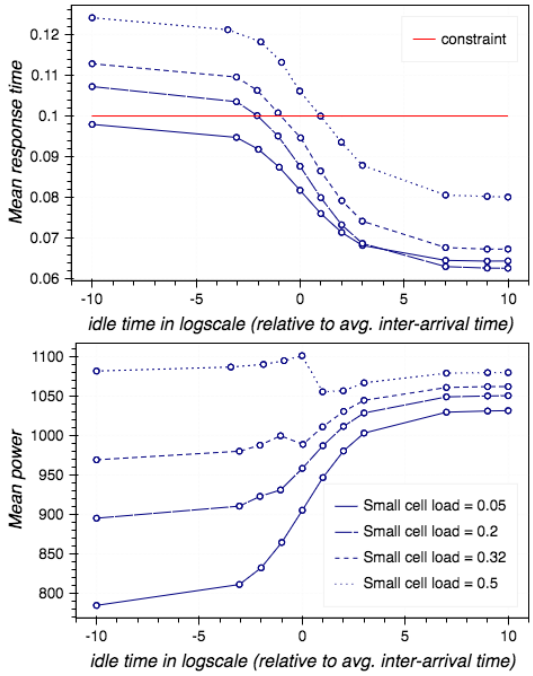


Fig. 4. Performance of FPI(β^*) policy as a function of mean idle timer with $E[D] = 0.05$. Mean idle timer (x-axis) is shown in base-2 logarithmic scale.

NEVEROFF or INSTANTOFF, with low load and short setup delay favoring INSTANTOFF. The results indicate that the general theoretical result that in a single energy-aware M/G/1-PS queue the optimal value of the idle timer is always either zero or infinity, see [17], applies also in this more complex setting without Poisson arrivals (FPI policy makes arrivals at each queue non-Poisson) for the ERWS cost function.

B. Constrained optimization

Figure 3 illustrates mean response time and mean power consumption of the system under $\pi^{\mathrm{FPI}}(\beta^*)$ policy as a function of average idle timer on small cells with $E[D] = 1$. The system behaves more intuitively with respect to mean response time, that is, the longer the cells are allowed to wait in idle state the shorter the response time will be on average. We also observe that largest improvements in mean response time are acheived for average idle timer values between 0.5 and 3 times the mean inter-arrival time. However, for a very light load (0.05 in the figure), longer idle timer has no effect as the policy puts all the traffic in the macro cell.

For points above the delay constraint (red solid line), the policy selects $\beta = 0$, as this is the best we can do to force mean response time as close as possible to the delay constraint under the given idle timer value. However, as soon as idle timer value is high enough so that the constraint is met, the policy starts to minimize mean power by selecting a β^* value different from zero, which explains the less steep decrease in mean response time near the constraint.

Now we focus on the power consumption plot in Figure 3. For the smallest load value of 0.05, the policy uses the macro cell only. In this case, mean power consumption is not affected by the choice of idle timer value, as long as it is finite, i.e., they are not configured as NEVEROFF. For load values 0.2, 0.32, and 0.5, the average power consumption decreases as idle timer increases. This looks counter-intuitive as shorter idle timers should enable the small cells to go to sleep more frequently, which should result in reduced average power. However, all three load values are too large to be handled by the macro cell alone, which means sleeping small cells will need to be started frequently. Idle power is avoided by putting cells to sleep, only to be followed by a higher power consumption in the setup state. The effect is amplified by the fact that setup delay is much longer than the average inter-arrival time (and hence the idle timer) resulting in a system that consumes more power than its energy-aware counterpart.

Therefore, for the given setup delay value, the optimal configuration is either INSTANTOFF or NEVEROFF. INSTANTOFF is optimal when load is low enough so that small cells can be switched off completely, whereas NEVEROFF is optimal in all other cases.

We further investigate the impact of setup delay by keeping all other parameters and considering a very short mean setup delay value of $E[D] = 0.05$, which is in the same order as the service time in the small cells. Figure 4 shows mean response time and mean power of such as system.

The mean response time curve roughly exhibits the same behavior as discussed above. However, mean power increases as a function of idle timer except for some discontinuities (explained below). With the setup delay being very short, the high setup power has less impact on the mean power

compared to the idle power, which makes the INSTANTOFF configuration of small cells an optimal choice for minimizing energy. However, looking back at the mean response time plot, the response time constraint enforces a non-zero idle timer for most of the load values. In this case, a DELAYEDOFF configuration will be optimal with the idle timer set to the smallest value that satisfies the response time constraint.

Notice the discontinuities in the mean power curves. When idle timer is too small, the mean response time does not meet the constraint, which leads to $\beta = 0$. But when the idle timer is long enough, so that the constraint is met with something to spare we start optimizing with respect to mean power, which explains the discontinuities. Note also that for the lowest load scenario, all idle timer values give feasible mean response times resulting in a smooth mean power curve.

VI. Conclusions

We have considered energy efficient load balancing in a system consisting of a macrocell with several small cells inside its coverage area. The system is modeled as a set of parallel queues consisting of a multiclass M/M/1-PS queue, representing the always-on macrocell, and each small cell is characterized by an energy-aware M/M/1-PS queue with a sleep state and setup delay. As an additional control feature, the model of the small cells included an idle timer, which is used for controlling how long the small cell waits after the end of a busy period until it falls into sleep state.

By applying the theory of MDPs and the first step of the policy iteration method, we developed a near optimal policy for the performance-energy trade-off. Our main contribution was the derivation of the explicit forms of the value functions for the energy and performance, which yields the FPI policy. The explicit form of the FPI policy yielded insights to the general propertied of the (near) optimal policy: the marginal performance cost has a JSQ-like linear dependence on the number of flows in the macrocell and small cell models and an additional constant cost in the small cell model reflecting the server state, while the marginal energy cost is constant both in the macrocell and small cell models. The FPI policy was initially derived by characterizing the performance-energy tradeoff as the weighted sum of performance and energy. We also showed how the same FPI policy can be applied through Lagrangian techniques in a constrained MDP setting, as well, where the objective is to minimize the energy subject to a performance constraint. In our numerical studies, we focused on the impact of the idle timer. In the case of the weighted sum objective function, the optimal timer value appears to be either zero or infinite. However, in the constrained optimization a finite idle timer can be optimal when the setup delay is short enough relative to the service times.

Possibilities for future research are many. On the algorithmic side, one research direction can be to seek for simple heuristic policies that achieve nearly the same performance as the FPI policy. Generalizations worth investigating include analyzing the impact of non-exponential service time distributions as well as interference between the base stations. However, these extensions are analytically very difficult to handle but simulations can be used to this end.

Acknowledgements

This research has been partially supported by EIT Digital under the HII ACTIVE project and by the Academy of Finland under the ITTECH5G project (Grant No. 284735).

References

[1] M. F. Hossain, K. S. Munasinghe, and A. Jamalipour, "Distributed inter-BS cooperation aided energy efficient load balancing for cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 11, pp. 5929–5939, 2013.

[2] E. Oh, K. Son, and B. Krishnamachari, "Dynamic base station switching-on/off strategies for green cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 5, pp. 2126–2136, 2013.

[3] J. Zheng, Y. Cai, X. Chen, R. Li, and H. Zhang, "Optimal base station sleeping in green cellular networks: A distributed cooperative framework based on game theory," *IEEE Transactions on Wireless Communications*, vol. 14, no. 8, pp. 4391–4406, 2015.

[4] S. Cai, Y. Che, L. Duan, J. Wang, S. Zhou, and R. Zhang, "Green 5G Heterogeneous Networks Through Dynamic Small-Cell Operation," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1103–1115, 2016.

[5] A. Rangisetti, T. Pasca, and B. R. Tamma, "QoS Aware load balance in software defined LTE networks," *Computer Communications*, vol. 97, pp. 52–71, 2017.

[6] G. Carvalho, I. Woungang, A. Anpalagan, and E. Hossain, "QoS-aware energy-efficient joint radio resource management in Multi-RAT heterogeneous networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 8, pp. 6343–6365, 2016.

[7] E. Khloussy, X. Gelabert, and Y. Jiang, "Investigation on MDP-based radio access technology selection in heterogeneous wireless networks," *Computer Networks*, vol. 91, pp. 57–67, 2015.

[8] A. Roy and A. Karandikar, "Optimal radio access technology selection policy for LTE-WiFi network," in *Proc. of WiOpt*, May 2015, pp. 291–298.

[9] Y. Song, P. Y. Kong, and Y. Han, "Potential of network energy saving through handover in HetNets," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 12, pp. 10 198–10 204, 2016.

[10] I. Taboada, S. Aalto, P. Lassila, and F. Liberal, "Delay- and energy-aware load balancing in ultra-dense heterogeneous 5G networks," *Transactions on Emerging Telecommunications Technologies*, vol. 28:e3170, 2017.

[11] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[12] M. Gebrehiwot, S. Aalto, and P. Lassila, "Near-optimal policies for energy-aware task assignment in server farms," in *Proc. of CCGrid*, May 2017, pp. 1017–1026.

[13] J. Leino and J. Virtamo, "Determining the moments of queue-length distribution of discriminatory processor-sharing systems with phase-type service requirements," in *Proc. of NGI*, May 2007, pp. 205–208.

[14] P. Osti, P. Lassila, and S. Aalto, "Optimal intercell coordination for multiple user classes with elastic traffic," in *Proc. of NGI*, June 2012, pp. 25–32.

[15] E. Altman, *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999.

[16] F. J. Beutler and K. W. Ross, "Optimal policies for controlled Markov chains with a constraint," *Journal of Mathematical Analysis and Application*, vol. 112, pp. 236–252, 1985.

[17] M. E. Gebrehiwot, S. Aalto, and P. Lassila, "Energy-performance trade-off for processor sharing queues with setup delay," *Operations Research Letters*, vol. 44, no. 1, pp. 101–106, 2016.

[18] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, pp. 1155–1171, 2010.

[19] C. Sun, E. Stevens-Navarro, V. Shah-Mansouri, and V. W. Wong, "A constrained MDP-based vertical handoff decision algorithm for 4G heterogeneous wireless networks," *Wireless Networks*, vol. 17, no. 4, pp. 1063–1081, May 2011.

Making Name-Based Content Routing More Efficient than Link-State Routing

Ehsan Hemmati* and J.J. Garcia-Luna-Aceves*,†
*Computer Engineering Department, UC Santa Cruz, Santa Cruz, CA 95064
†PARC, Palo Alto, CA 94304
{ ehsan, jj }@soe.ucsc.edu

***Abstract*—The Diffusive Name-based Routing Protocol (DNRP) is introduced for efficient name-based routing in information-centric networks (ICN). DNRP establishes and maintains multiple loop-free routes to the nearest instances of a name prefix using only distance information. DNRP eliminates the need for periodic updates, maintaining topology information, storing complete paths to content replicas, or knowing about all the sites storing replicas of named content. DNRP is suitable for large ICNs with large numbers of prefixes stored at multiple sites. It is shown that DNRP provides loop-free routes to content independently of the state of the topology, and that it converges within a finite time to correct routes to name prefixes after arbitrary changes in the network topology or the placement of prefix instances. The result of simulation experiments illustrate that DNRP is more efficient than link-state routing approaches.**

I. Introduction

Several Information-Centric Networking (ICN) architectures have been introduced to address the increasing demand of user-generated content [1], [3]. The goal of these architectures is to provide a cost-efficient, scalable, and mobile access to content and services by adopting a content-based model of communication. ICN architectures seek to dissociate content and services from their producers in such a way that the content can be retrieved independently of its original location or the location of consumers. The most prominent ICN architectures can be characterized as Interest-based architectures, in which location-independent, self-defined, unique naming is used to retrieve data. In this approach, messages flow from producers to consumers based on the name of the content rather than the address of the senders or receivers exchanging such content. Content providers or producers create named data objects (NDOs), and advertise routable name prefixes associated with the content objects whose own names are part of the name prefixes. The only identifier of an NDO is its name. A consumer requests a piece of content by sending an Interest (a request for the NDO) that is routed along content routers toward the producer(s).

Clearly, an efficient name-based content routing protocol must be used for any ICN architecture to succeed using name-based forwarding of Interests and requested content. Section II summarizes recent prior work in name-based content routing, and this review reveals that all prior proposals for name-based content routing rely on periodic transmissions of update messages. This paper focuses on an approach that avoids the need for periodic messaging by means of diffusing computations [5].

ISBN 978-3-903176-08-9

Section III presents **DNRP** (*Diffusive Name-based Routing Protocol*), a name-based content routing protocol for ICNs. DNRP provides multiple loop-free routes to the nearest instances of a named prefix or to all instances of a named prefix using only distance information and without requiring periodic updates, knowledge of the network topology, or the exchange of path information.

Section IV shows that DNRP prevents routing-table loops even in the presence of topology changes as well as changes in the hosting of prefixes, and converges within a finite time to correct multi-paths to name prefixes. Section V presents the results of simulation experiments comparing DNRP with an efficient link-state approach similar to NLSR [13]. The results show that DNRP produces less communication and computation overhead in the case of topology changes as well as the addition of prefixes.

II. Related Work

Name-based content routing has been used in the past in content-delivery networks (CDN) operating as overlay networks running on top of the Internet routing infrastructure (e.g., [7], [16]). However, it has become more well known in the context of ICN architectures, where it has been done typically by adapting traditional routing algorithms designed for networks in which a destination has a single instance [8]. Distributed Hash Tables (DHT) are used in several architectures as the name resolution tool [11], [15], [18]. MobilityFirst [14] rely on an external and fast name resolution system called Global Name Resolution Service (GNRS) that maps the data object names to network addresses.

Some ICN architectures use name resolution mechanisms to map the name of the content to the content provider. Data Oriented Network Architecture (DONA) [12] replaces DNS names with “flat, self-certifying names” and uses name resolution to map those flat names to corresponding IP addresses. DONA supports host mobility and multihoming, and improves service access and data retrieval.

The Named-data Link State Routing protocol (NLSR) [13] uses link state routing to rank the neighbors of a router for each name prefix. ”Adjacency LSA” and ”Prefix LSA”, propagate topology and publisher information in the network respectively. Each router uses topology information and runs

an extension of Dijkstra's shortest-path first (SPF) algorithm to rank next hops for each router, then maps the prefix to the name of the publisher and creates routing table for each name prefix. Like most prior routing approaches based on complete or partial topology information (e.g., [2], [6], [17]), NLSR uses sequence numbers to allow routers to determine whether the updates they receive have more recent information than the data they currently store. As a consequence, these approaches require the use of periodic updates to percolate throughout the network to ensure that all routers converge to consistent topology information within a finite time.

The Distance-based Content Routing (DCR) [10] was the first name-based content routing approach for ICNs based on distances to named content. DCR does not require any information about the network topology or knowledge about all the instances of the same content. It enables routing to the nearest router announcing content from a name prefix being stored locally (called anchor), all anchors of a name prefix, and subsets of anchors. This is attained by means of multi-instantiated destination spanning trees (MIDST). Furthermore, DCR provides loop-free routes to reach any piece of named content even if different content objects in the same prefix are stored at different sites. The limitation of DCR is that it requires periodic updates to be disseminated through the network.

III. DNRP

DNRP finds the shortest path(s) to the nearest replica(s) of name prefixes. To ensure that loop-free routes to named prefixes are maintained at every instant independently of the state of the network or prefixes, DNRP establishes a lexicographic ordering among the routes to prefixes reported and maintained by routers. The lexicographic ordering of routes is based on two sufficient conditions for loop freedom with respect to a given prefix that allow for multiple next hops to prefixes along loop-free routes. DNRP diffuses the computation of new loop-free routes when the loop-free conditions are not satisfied. The approach used in DNRP is an extrapolation to the use of diffusing computations in [5].

Every piece of data in the network is a *Named-Data Object* (*NDO*), represented by a name that belongs to a name prefix or simply a *prefix* advertised by one or more producer(s). Name prefixes can be simple and human-readable or more complicated and self certifying, or may even be a cryptographic hash of the content. Content names can be flat or hierarchical. The naming schema depends on the system that runs DNRP and it is out of scope of this paper.

A router attached to a producer of content that advertises a name prefix is called an $anchor$ of that prefix. At each router, DNRP calculates routes to the nearest anchor(s) of known name prefixes, if there is any, and selects a subset of the neighbors of the router as valid next hops to reach name prefixes, such that no routing-table loop is created at any router for any name prefix. Caching sites are not considered content producers and hence routes to cached content are not advertised in DNRP. Our description assumes that routers process, store, and transfer information correctly and that they process routing messages one at a time within a finite time. Every router has a unique identifier or a name that can be flat or hierarchical. The naming schema and name assignment mechanism is out of scope of this paper.

A. Messages and Data Structures

Each router i stores the list of all active neighbor routers (N^i), and the cost of the link from the router to each such neighbor. The cost of the link from router i to its neighbor n is denoted by l^i_n. Link costs can vary in time but are always positive. The link cost assignment and metric determination mechanisms are beyond the scope of this paper.

The routing information reported by each of the neighbors of router i is stored in its *neighbor table* (NT^i). The entry of NT^i regarding neighbor n for prefix p is denoted by NT^i_{pn} and consists of the name prefix (p), the distance to prefix p reported by neighbor n (d^i_{pn}), and the anchor of that prefix reported by neighbor n (a^i_{pn}). If router i is the anchor of prefix p itself, then d^i_{pi} = 0.

Router i stores routing information for each known prefix in its routing table (RT^i). The entry in RT^i for prefix p (RT^i_p) specifies: the name of the prefix (p); the distance to the nearest instance of that prefix (d^i_p); the feasible distance to the prefix (fd^i_p); the neighbor that offers the shortest distance to the prefix (s^i_p), which we call the successor of the prefix; the closest anchor to the prefix (a^i_p); the state mode (mf^i_p) regarding prefix p, which can be *PASSIVE* or *ACTIVE*; the origin state (o^i_p) indicating whether router i or a neighbor is the origin of query in which the router is active; the update flag list (FL^i_p); and the list of all valid next hops (V^i_p).

FL^i_p consists of four flags for each neighbor n. An update flag (uf^i_{pn}) denotes whether or not the routing message should be sent to that neighbor. A type flag (tf^i_{pn}) indicates the type of routing message the router has to send to neighbor n regarding prefix p (i.e., whether it is an *UPDATE*, *QUERY*, or *REPLY*). A pending-reply flag (rf^i_{pn}) denotes whether the router has sent a *QUERY* to that neighbor and is waiting for *REPLY*. A pending-query flag (qf^i_{pn}) is set if the router received a *QUERY* from its neighbor n and has not responded to that *QUERY* yet.

Router i sends a routing message to each of its neighbors containing updates made to RT^i since the time it sent its last update message. A routing message from router i to neighbor n consists of one or more updates, each of which carries information regarding one prefix that needs updating. The update information for prefix p is denoted by U^i_p and states: (a) the name of the prefix (p); (b) the message type (ut^i_p) that indicates if the message is an *UPDATE*, *QUERY*, or *REPLY*; (c) the distance to p; and (d) the name of the closest anchor.

The routing update received by router i from neighbor n is denoted by U^i_n. The update information of U^i_n for prefix p, u^i_{pn}, specifies the prefix name (p), the distance from n to that prefix (ud^i_{pn}), the name of the anchor of that prefix (ua^i_{pn}), and a message type (ut^i_{pn}).

B. Sufficient Conditions for Loop Freedom

The conditions for loop-free routing in DNRP are based on the feasible distance maintained at each router and the distances reported by its neighbors for a name prefix. One condition is used to determine the new shortest distance through a loop-free path to a name prefix. The other is used to select a subset of neighbors as next hops to a name prefix.

Source Router Condition (SRC): Router i can select neighbor $n \in N^i$ as a new successor s_p^i for prefix p if:

$$(d_{pn}^i < \infty) \wedge (d_{pn}^i < fd_p^i \vee [d_{pn}^i = fd_p^i \wedge |n| < |i|]) \wedge (d_{pn}^i + l_n^i = Min\{d_{pv}^i + l_v^i | v \in N^i\}). \square$$

SRC simply states that router i can select neighbor n as its successor to prefix p if n reports a finite distance to that prefix, offers the smallest distance to prefix p among all neighbors, and either its distance to prefix p is less than the feasible distance of router i or its distance is equal to the feasible distance of i but $|n| < |i|$. If two or more neighbors satisfy *SRC*, the neighbor that satisfies *SRC* and has the smallest identifier is selected. If none of the neighbors satisfies *SRC* the router keeps the current successor, if it has any. The distance of router i to prefix, d_p^i, is defined by the distance of the path through the selected successor.

A router that has a finite feasible distance ($fd_p^i < \infty$) selects a subset of neighbors as valid next hops at time t if they have a finite distance to destination and are closer to destination. The following condition is used for this purpose.

Next-hop Selection Condition (NSC): Router i with $fd_p^i < \infty$ adds neighbor $n \in N^i$ to the set of valid next hops if: $(d_{pn}^i < \infty) \wedge (d_{pn}^i < fd_p^i \vee [d_{pn}^i = fd_p^i \wedge |n| < |i|]). \square$

NSC states that router i can select neighbor n as a next hop to prefix p if either the distance from n to prefix p is smaller than the feasible distance of i or its distance is equal to the feasible distance and $|n| < |i|$. *NSC* orders next hops lexicographically based on their distance to a prefix and their names. It is shown that no routing-table loops can be formed if *NSC* is used to select the next hops to prefixes at each router. Note that the successor is also a valid next hop. The successor to a prefix is a valid next hop that offers the smallest cost.

SRC and *NSC* are *sufficient conditions* that, as we show subsequently, ensure loop-freedom at every instance but do not guarantee shortest paths to destinations. DNRP integrates these sufficient conditions with inter-nodal synchronization signaling to achieve both loop freedom at every instant and shortest paths for each destination.

C. DNRP Operation

A change in the network, such as a link-cost change, the addition or failure of a link, the addition or failure of a router, the addition or deletion of a prefix, or the addition or deletion of a replica of a prefix can cause one or more computations at each router for one or more prefixes. A computation can be either a *local computation* or a *diffusing computation*. In a local computation a router updates its successor, distance, next hops, and feasible distance independently of other routers in the network. On the other hand, in a diffusing computation a router originating the computation must coordinate with other routers before making any changes in its routing-table entry for a given prefix. DNRP allows a router to participate in at most one diffusing computation per prefix at any given time.

A router can be in *PASSIVE* or *ACTIVE* mode with respect to a given prefix independently of other prefixes. A router is *PASSIVE* with respect to prefix p if it is not engaged in any diffusing computation regarding that prefix. A router initializes itself in *PASSIVE* mode and with a zero distance to all the prefixes for which the router itself serves as an anchor. An infinite distance is assumed to any non-local (and hence unknown) name prefix.

Initially, no router is engaged in a diffusing computation ($o_p^i = 0$). When a *PASSIVE* router detects a change in a link or receives a *QUERY* or *UPDATE* from its neighbor that does not affect the current successor or can find a feasible successor, it remains in *PASSIVE* mode. On the other hand, if the router cannot find a feasible successor then it enters the *ACTIVE* mode and keeps the current successor, updates its distance, and sends *QUERY* to all its neighbors. Table I shows the transit from one state to another. Neighbor k is a neighbor other than the successor s.

TABLE I
STATE TRANSIT IN DNRP

Mode	*State*	Event	*Next State*
Passive	0	Events from a neighbor k, *SRC* satisfied	0
		Events from a neighbor k, *SRC* not satisfied	1
		QUERY from the *Successor*	3
Active	1	Receives last *REPLY*	0
		Change in distance to *Successor*	2
		QUERY from the *Successor*	4
	2	Receives last *REPLY*, *SRC* satisfied	0
		Receives last *REPLY*, *SRC* not satisfied	1
		QUERY from the *Successor*	4
	3	Receives last *REPLY*	0
		Change in distance to the *Successor*	4
	4	Receives last *REPLY*, *SRC* satisfied	0
		Receives last *REPLY*, *SRC* not satisfied	3

Algorithm 1 shows the processing of messages by a router in *PASSIVE* mode. Algorithm 2 shows the steps taken in *ACTIVE* mode. Algorithm 3 shows the steps taken to process a routing update.

Handling A Single Diffusing Computation: Routers are initialized in *PASSIVE* mode. Each router continuously monitors its links and processes the routing messages received from its neighbors. When router i detects a change in the cost or state of a link, or a change in its neighbor table that causes a change in its distance to prefix p (d_p^i), it first tries to select a new successor that satisfies *SRC*. If such a successor exists, the router carries out a local computation, updates its distance, successor, and closest anchor, and exits the computation. In a local computation, router i computes the minimum cost to reach the destination and updates $d_p^i = min\{d_{pn}^i + l_n^i | n \in$

$N^i\}$. If its distance changes, router i sends a routing message with $ut_p^i = UPDATE$. Router i also updates its feasible distance to equal the smaller of its value and the new distance value, i.e., $fd_p^i(new) = min\{fd_p^i(old), d_p^i\}$.

An *UPDATE* message from a neighbor is processed using the same approach stated above. If a router receives a *QUERY* from its neighbor other than its successor while it is in *PASSIVE* mode, it updates the neighbor table, checks for a feasible successor according to *SRC* and replies with d_p^i, if it succeeds. If router i cannot find a neighbor that satisfies *SRC* after a change in a link or neighbor-table entry, then it starts out a diffusing computation by setting the new distance as the distance through its current successor, enters the *ACTIVE* mode ($mf_j^i =$ *ACTIVE*) and sets the corresponding flag ($rf_p^i n$) for each neighbor n. After entering the *ACTIVE* mode, router i sets the new distance as the cost of the path through the current successor ($d_p^i = d_{ps_p^i}^i + l_{s_p^i}^i$) and sends a routing message with $ut_p^i =$ *QUERY*. Router i uses the pending reply flag (rf_{pn}^i) to keep track of the neighbors from which a *REPLY* has not been received. When a router becomes *ACTIVE* it sets the update flag ($uf_{pn}^i = 1$), and also sets the type flags ($tf_{pn}^i = QUERY | \forall n \in N^i$) and sends the routing messages to all its neighbors.

Algorithm 1 Processing routing messages in *PASSIVE* mode

INPUT: $RT^i, NT^i, l_n^i, u_{pn}^i$;
[o] verify u_{pn}^i;
$d_{pn}^i = ud_{pn}^i$; $d_{min} = \infty$;
for each $k \in N^i - \{i\}$ **do**
  if $(d_{pk}^i + l_k^i < d_{min}) \vee (d_{pk}^i + l_k^i = d_{min} \wedge |k| < |s_{new}|)$ **then**
    $s_{new} = k; d_{min} = d_{pk}^i + l_k^i$;
  end if
end for
if $(d_{ps_{new}}^i < fd_p^i \vee [d_{ps_{new}}^i = fd_p^i \wedge |s_{new}| < |i|])$ **then**
  if $s_p^i \neq s_{new}$ **then** $s_p^i = s_{new}$; $a_p^i = a_{ps_{new}}^i$
  if $d_p^i \neq d_{min}$ **then**
    $d_p^i = d_{min}$; $fd_p^i = min\{fd_p^i, d_p^i\}$; $V_p^i = \phi$;
    for each $k \in N^i - \{i\}$ **do**
      $uf_{pk}^i = 1$; $tf_{pk}^i =$*UPDATE*;
      if $(d_{kp}^i < fd_p^i \vee [d_{pk}^i = fd_p^i \wedge |k| < |i|])$ **then**
        $V_p^i = V_p^i \cup k$;
      end if
    end for
    if $ut_{pn}^i =$ *QUERY* **then** $tf_{pn}^i =$*REPLY*;
  end if
else
  $mf_p^i =$ *ACTIVE*; $d_p^i = d_{ps_p^i}^i + l_{s_p^i}^i$;
  if $(n = s_p^i \wedge ut_{pn}^i =$ *QUERY*$)$ **then** $o_p^i = 3$; **else** $o_p^i = 1$;
  for each $k \in N^i - \{i\}$ **do** $uf_{pn}^i = 1$; $tf_{pn}^i =$*QUERY*;
end if

When a router is in *ACTIVE* mode, it cannot change its successor or fd_p^i until it receives the replies to its *QUERY* from all its neighbors. After receiving all replies (i.e. $rf_{pn}^i = 0 | \forall n \in N^i$), router i becomes *PASSIVE* by resetting its feasible distance. The router then selects the new successor and sends *UPDATE* messages to its neighbors. More specifically, router i sets $fd_p^i = \infty$ which insures that the router can find a new successor that satisfies *SRC* and then sets $fd_p^i = d_p^i = min\{d_{pn}^i + l_n^i | n \in N^i\}$ and becomes *PASSIVE*.

Algorithm 2 Processing routing messages in *ACTIVE* mode

INPUT: RT^i, NT^i, u_{pn}^i;
[o] verify u_{pn}^i;
$d_{pn}^i = ud_{pn}^i$;
if $ut_{pn}^i =$ *REPLY* **then**
  $rf_{pn}^i = 0; lastReply = true$;
  for each $k \in N^i - \{i\}$ **do**
    if $rf_{pk}^i = 0$ **then** $lastReply = false$;
  end for
  if $lastReply = true$ **then**
    if $o_p^i = 1 \vee o_p^i = 3$ **then** $fd_p^i = \infty$
    Execute Algorithm 3
  end if
else if $ut_{pn}^i =$ *QUERY* **then**
  if $(o_p^i = 1 \vee o_p^i = 2)$ **then**
    if $n \neq s_p^i$ **then** $uf_{pn}^i = 1$; $tf_{pn}^i =$*REPLY;* **else** $o_p^i = 4$;
  end if
  if $(o_p^i = 3 \vee o_p^i = 4)$ **then** $uf_{pn}^i = 1$; $tf_{pn}^i =$*REPLY;*
end if

Algorithm 3 Update RT_p^i

INPUT: $RT^i, NT^i, l_n^i, u_{pn}^i$;
$d_{min} = \infty$;
for each $k \in N^i - \{i\}$ **do**
  if $(d_{pk}^i + l_k^i < d_{min}) \vee (d_{pk}^i + l_k^i = d_{min} \wedge |k| < |s_{new}|)$ **then**
    $s_{new} = k; d_{min} = d_{pk}^i + l_k^i$;
  end if
end for
if $(d_{ps_{new}}^i < fd_p^i \vee [d_{ps_{new}}^i = fd_p^i \wedge |s_{new}| < |i|])$ **then**
  $o_p^i = 0; mf_p^i = PASSIVE$;
  if $s_p^i \neq s_{new}$ **then** $s_p^i = s_{new}$;
  if $d_p^i \neq d_{min}$ **then**
    $d_p^i = d_{min}; fd_p^i = min\{fd_p^i, d_p^i\}; V_p^i = \phi$;
    for each $k \in N^i - \{i\}$ **do**
      $uf_{pk}^i = 1$; $tf_{pk}^i =$*UPDATE*;
      if $(d_{pk}^i < fd_p^i \vee [d_{pk}^i = fd_p^i \wedge |k| < |i|])$ **then**
        $V_p^i = V_p^i \cup k$;
      end if
    end for
    if $qf_{ps_p^i}^i(old) = 1$ **then** $tf_{pn}^i =$*REPLY;*
  end if
else
  if $o_p^i = 2$ **then** $o_p^i = 1$ **else** $o_p^i = 3$;
  for each $k \in N^i - \{i\}$ **do** $uf_{pn}^i = 1$; $tf_{pn}^i =$*QUERY*;
end if

If router i receives a *QUERY* from a neighbor other than its successor while it is *ACTIVE*, it simply replies to its neighbor with a *REPLY* message stating the current distance to the destination. The case of a router receiving a *QUERY* from its successor while it is *ACTIVE* is described subsequently in the context of multiple diffusing computations. *UPDATE* messages are processed and neighbor tables are updated, but the successor or distance is not changed until the router receives all the replies it needs to transition to the *PASSIVE* mode. While a router is in *ACTIVE* mode, neither a *QUERY* nor an *UPDATE* can be sent.

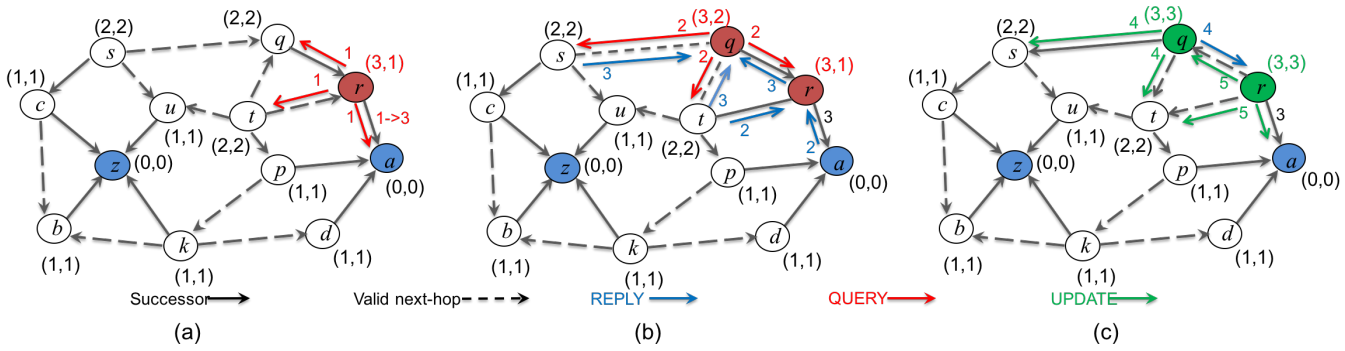


Fig. 1. DNRP Operation Example

Handling Multiple Diffusing Computations: Given that a router executes each local computation to completion, it handles multiple local computations for the same prefix one at a time. Similarly, a router handles multiple diffusing computation for the same prefix by processing one computation at a time. An *ACTIVE* router i can be in one of the following four states: (1) router i originated a diffusing computation ($o_p^i = 1$), (2) metric increase detected during *ACTIVE* mode ($o_p^i = 2$), (3) diffusing computation is relayed ($o_p^i = 3$), or (4) successor metric changed during *ACTIVE* mode ($o_p^i = 4$). If the router is in *PASSIVE* mode then its state is 0 (i.e., $o_p^i = 0$).

Consider the case that a router i is *ACTIVE* and in State 1 ($o_p^i = 1$). If the router receives the last *REPLY* to its query, then it resets its feasible distance to infinity, checks *SRC* to find the new successor, and sends an *UPDATE* to all its neighbors. On the other hand, if router i detects a change in the link to its successor then it updates its neighbor table and sets $o_p^i = 2$.

If router i is in State 2, receives the last *REPLY*, and can find a feasible successor using *SRC* with the current feasible distance, then it becomes *PASSIVE* and sends an *UPDATE* to all its neighbors($o_p^i = 0$). Otherwise, it sends a *QUERY* with the current distance and sets $o_p^i = 1$.

Router i uses the pending query flag (qf_{pn}^i) to keep track of the replies that have been received for its *QUERY* regarding prefix p. If router i is in either State 1 or 2 and receives a *QUERY* from its current successor to the prefix, then it sets $qf_{ps_p^i=1}^i$ and transitions to State 4 (i.e., it sets $o_p^i = 4$).

If a router in *PASSIVE* mode receives a *QUERY* from its successor, it searches for a new successor that satisfies *SRC*. If it cannot find such a successor then it keeps the current successor, updates its distance, and becomes *ACTIVE*. Then, the router sends *QUERY* to all of its neighbors and sets $o_p^i = 3$.

When router i in state 3 receives *REPLY* from all of its neighbors, it resets its feasible distance, $fd_p^i = \infty$, selects a new successor, updates the V_p^i and sends *UPDATE* to its neighbors and *REPLY* to its the previous successor. If the router detects a link failure or a cost increase in the link to its current successor, the router sets $o_p^i = 4$ to indicate that a topology change occurred while the router is in *ACTIVE* mode. A router handles the case of the failure of the link with its successor as if it had received a *REPLY* from its successor with $d_{ps_p^i}^i = \infty$.

If router i is in State 4, ($o_p^i = 4$) and it receives replies from all its neighbors, then it tries to find a feasible successor that satisfies *SRC* with the current value of fd_p^i. If such a successor exists, the router updates its successor, distance, and next hops for prefix p, and sends an *UPDATE* message to its neighbors as well as *REPLY* to the previous successor. Otherwise, it sets $o_p^i = 3$ and sends a *QUERY* with the new distance.

While router i is in *ACTIVE* mode regarding a prefix, if a *QUERY* is received for the prefix from a neighbor other than the current successor, the router updates the neighbor table and sends a *REPLY* to that neighbor. If a router in *PASSIVE* mode receives a *QUERY* from a neighbor other than the current successor, the router updates its neighbor table. If the feasibility condition is not satisfied anymore, the router sends a *REPLY* to the neighbor that provides the current value d_p^i before it starts its own computation.

D. Example of DNRP Operation

Figure 1 illustrates the operation of DNRP with a simple example. The figure shows the routing information used for a single prefix when routers a and z advertise that prefix and each link has unit cost. The tuple next to each router states the $distance$ and the $feasible\ distance$ of the router for that prefix. The red, blue, and green arrows represent the *QUERY*, *REPLY*, and *UPDATE* messages respectively and the number next to the arrow shows the time sequence in which that message is sent. Figure 1 (a) shows the change in the cost of link (r, a). Router r detects this change and becomes *ACTIVE* and sends *QUERY* to its neighbors.

Router q receives the *QUERY* from its successor and cannot find a feasible successor (Figure 1(b)). Therefore, it becomes *ACTIVE* and sends a *QUERY* to its neighbors. Router r receives *REPLY* from a and t, and a *QUERY* from q. Given that q is not a successor for router r, r sends *REPLY* to q. After receiving *REPLY* from routers r, s and t, router q becomes *PASSIVE* again and sends its *REPLY* to its previous successor, r. In turn, this means that r receives all the replies it needs, becomes *PASSIVE*, and resets its feasible distance. The operation of DNRP is such that only a portion of the routers are affected by the topology change.

E. Routing to all instances of a prefix

DNRP enables routers to maintain multiple loop-free routes to the nearest anchor of a name prefix. In some ICN architectures, such as NDN and CCNx, an anchor of a name-prefix may have some but not necessarily all the content corresponding to a given prefix. Therefore, simply routing to nearest replica may cause some data to be unreachable, and the ability to contact all anchors of a prefix is needed. To address this case, a multi-instantiated destination spanning tree (*MIDST*) can be used alongside DNRP to support routing to all anchors of the same prefix. A MIDST is established in a distributed manner. Routers that are aware of multiple anchors for the same prefix exchange routing updates to establish the spanning tree between all anchors of a prefix. Once the *MIDST* is formed for a given prefix, the first router in the MIDST that receives a packet forwards it over the *MIDST* to all of the anchors. The details of how a *MIDST* can be established in DNRP are omitted for brevity; however, the approach is very much the same as that described in [9].

IV. Correctness of DNRP

The following theorems prove that DNRP is loop-free at every instant and considers each computation individually and in the proper sequence. From these results, the proof that DNRP converges to shortest paths to prefixes is similar to the proof presented in [4] and due to space limitation is omitted. We assume that each router receives and processes all routing messages correctly. This implies that each router processes messages from each of its neighbors in the correct order.

Theorem 1: No routing-table loops can form in a network in which routers use *NSC* to select their next hops to prefixes.

Proof: Assume for the sake of contradiction that all routing tables are loop-free before time t_l but a routing-table loop is formed for prefix p at time t_l when router q adds its neighbor n_1 to its valid next-hop set V_p^q. Because the successor is also a valid next hop, router q must either choose a new successor or add a new neighbor other than its current successor to its valid next-hop set at time t_l. We must show that the existence of a routing-table loop is a contradiction in either case.

Let L_p be the routing-table loop consisting of h hops starting at router q, ($L_p = \{q = n_{0,new}, n_{1,new}, n_{2,new}, \ldots, n_{h,new}\}$) where $n_{h,new} = q$, $n_{i+1,new} \in V_p^{n_i}$ for $0 \leq i \leq h$.

The time router n_i updates its valid next-hop set to include $n_{i+1,new}$ is denoted by t^i_{new}. Assume that the last time router n_i sent an *UPDATE* that was processed by its neighbor n_{i-1}, is t^i_{old}. Router n_i revisits valid next hops after any changes in its successor, distance, or feasible distance; therefore, $t^i_{old} \leq t^i_{new} \leq t_l$ and $d^{n_i}_{pn_{i+1}}(t_l) = d^{n_i}_{pn_{i+1}}(t_{old})$. Also, by definition, at any time t_i, $fd^i_p(t_i) \leq d^i_p(t_i)$, and $fd^i_p(t_2) \leq fd^i_p(t_1)$ if $t_1 < t_2$. Therefore,

$$fd^i_p(t_2) \leq d^i_p(t_1) \text{ such that } \; t_1 < t_2 \tag{1}$$

If router n_i selects a new successor at time t^i_{new} then:

$$d^{n_{i-1}}_{pn_i}(t_l) = d^{n_i}_p(t_{old}) \geq fd^{n_i}_p(t_{old}) \geq fd^{n_i}_p(t_{new}) \tag{2}$$

Using *NSC* ensures that

$$\begin{aligned}(fd^{n_i}_p(t_{new}) > d^{n_i}_{pn_{i+1}}(t_l))\\ \vee(fd^{n_i}_p(t_{new}) = d^{n_i}_{pn_{i+1}}(t_l) \wedge |n_i| > |n_{i+1}|)\end{aligned} \tag{3}$$

From Eqs. (2) and (3) we have:

$$\begin{aligned}(d^{n_{i-1}}_{pn_i}(t_l) > d^{n_i}_{pn_{i+1}}(t_l))\\ \vee(d^{n_{i-1}}_{pn_i}(t_l) = d^{n_i}_{pn_{i+1}}(t_l) \wedge |n_i| > |n_{i+1}|)\end{aligned} \tag{4}$$

Therefore, for $0 \leq k \leq h$ in L_p it is true that:

$$\begin{aligned}(d^{n_0}_{pn_1}(t_l) > d^{n_k}_{pn_{k+1}}(t_l))\\ \vee(d^{n_0}_{pn_1}(t_l) = d^{n_k}_{pn_{k+1}}(t_l) \wedge |n_0| > |n_k|)\end{aligned} \tag{5}$$

If $d^{n_{i-1}}_{pn_i}(t_l) > d^{n_i}_{pn_{i+1}}(t_l)$ in at least one hop in L_p then it must be true that, for any given $k \in \{1, 2, \ldots, h\}$, $d^{n_k}_{pn_{k+1}}(t_l) > d^{n_k}_{pn_{k+1}}(t_l)$, which is a contradiction. If at any hop in the L_p it is true that $d^{n_{i-1}}_{pn_i}(t_l) = d^{n_i}_{pn_{i+1}}(t_l)$, then $|k| > |k|$, which is also a contradiction. Therefore, no routing-table loop can be formed when routers use *NSC* to select their next hops to prefix p. ■

Lemma 2: A router that is not the origin of a diffusing computation sends a *REPLY* to its successor when it becomes *PASSIVE*.

Proof: A router that runs DNRP can be in either *PASSIVE* or *ACTIVE* mode for a prefix p when it receives a *QUERY* from its successor regarding the prefix. Assume that router i is in *PASSIVE* mode when it receives a *QUERY* from its successor. If router i finds a neighbor that satisfies *SRC*, then it sets its new successor and sends a *REPLY* to its old successor. Otherwise, it becomes *ACTIVE*, sets $o^i_p = 3$, and sends a *QUERY* to all its neighbors. Router i cannot receive a subsequent *QUERY* from its successor regarding the same prefix, until it sends a *REPLY* back to its successor. If the distance does not increase while router i is *ACTIVE* then o^i_p remains the same (i.e. $o^i_p = 3$). Otherwise, router i must set $o^i_p = 4$. In both cases router i must send a *REPLY* when it becomes *PASSIVE*.

Assume that router i is in *ACTIVE* mode when it receives a *QUERY* from its successor s. Router s cannot send another *QUERY* until it receives a *REPLY* from all its neighbors to its query, including router i. Hence, router i must be the origin of the diffusing computation for which it is *ACTIVE* when it receives the *QUERY* from s, which means that $o^i_p = 1$ or $o^i_p = 2$. In both cases router i sets $o^i_p = 4$ when it receives a *QUERY* form its successor s and s must send a *REPLY* in response to the *QUERY* from i because, i is not the successor for s. After receiving the last *REPLY* from its neighbors, either router i finds a feasible successor and sends a *REPLY* to s ($o^i_p = 0$) or it propagates the diffusing computation forwarded by s by sending a *QUERY* to its neighbors and setting $o^i_p = 3$. Router i then must send a *REPLY* to s when it receives the last *REPLY* for the *QUERY* it forwarded from s.

Hence, independently of its current mode, router i must send a *REPLY* to a *QUERY* it receives from its successor when it becomes *PASSIVE*. ■

Lemma 3: Consider a network that is loop free before an arbitrary time t and in which a single diffusing computation takes place. If node n_i is *PASSIVE* for prefix p at that time, then it must be true that $(d_{pn_i}^{n_{i-1}}(t) > d_{pn_{i+1}}^{n_i}(t)) \vee (d_{pn_i}^{n_{i-1}}(t) = d_{pn_{i+1}}^{n_i}(t) \wedge |n_i| > |n_{i+1}|)$ independently of the state of other routers in the chain of valid next hops $\{n_{i-1}, n_i, n_{i+1}\}$ for prefix p.

Proof: Assume that router n_i is *PASSIVE* and selects router n_{i+1} as a valid next hop. According to *NSC* it must be true that:

$$\begin{array}{c}(d_{pn_{i+1}}^{n_i}(t) < fd_p^{n_i}(t) \leq d_p^{n_i}(t)) \vee \\ (d_{pn_{i+1}}^{n_i}(t) = fd_p^{n_i}(t) \leq d_p^{n_i}(t) \wedge |n_{i+1}| < |n_i|)\end{array} \quad (6)$$

Assume that n_i did not reset $fd_p^{n_i}$ the last time $t_{new} < t$ when n_i became *PASSIVE* and selected its successor s_{new} and updated its distance $d_p^{n_i}(t_{new}) = d_p^{n_i}(t)$. If router n_{i-1} processed the message that router n_i sent after updating its distance, then: $d_{pn_i}^{n_{i-1}}(t) = d_p^{n_i}(t_{new})$. Substituting this equation in 6 renders the result of this lemma.

On the other hand, If router n_{i-1} did not process the message that router n_i sent after updating its distance and before t, then $d_{pn_i}^{n_{i-1}}(t) = d_p^{n_i}(t_{old})$. Based on the facts that router n_i did not reset its feasible distance and Eq. 1 holds for this case. Therefore:

$$d_{pn_i}^{n_{i-1}}(t) = d_{pn_i}^{n_{i-1}}(t_{old}) > fd_p^{n_i}(t) \quad (7)$$

Now consider the case that n_i becomes *PASSIVE* at time t_{new} and changes its successor from s_{old} to s_{new} by reseting its feasible distance. The case that n_{i-1} processed the message that router n_i sent after becoming *PASSIVE* is the same as before. Assume that n_{i-1} did not process the message that n_i sent at time t_{new}. Furthermore, assume that router n_i becomes *ACTIVE* at time t_{old}, with a distance $d_p^{n_i}(t_{old}) = d_{ps_{old}}^{n_i} + l_{s_{old}}^{n_i}$. Router n_i cannot change its successor or experience any increment in its distance through s_{old}; hence, $d_p^{n_i}(t_{new}) \leq d_p^{n_i}(t_{old})$. On the other hand, the distance through the new successor must be the shortest and so $d_p^{n_i}(t_{new}) = d_{ps_{new}}^{n_i} + l_{s_{new}}^{n_i} \leq d_p^{n_i}(t_{old})$. Router n_i becomes *PASSIVE* if it receives a *REPLY* from each of its neighbors including n_{i-1}. Therefore, n_{i-1} must be notified about $d_p^{n_i}(t_{old})$. Therefore:

$$d_{pn_i}^{n_{i-1}}(t) = d_p^{n_i}(t_{old}) \geq d_p^{n_i}(t_{new}) = d_p^{n_i}(t). \quad (8)$$

Substituting this equation in 6 renders the result of this lemma. Therefore, the lemma is true in all cases. ■

Lemma 4: Consider a network that is loop free before an arbitrary time t and in which a single diffusing computation takes place. Let two network nodes n_i and n_{i+1} be such that $n_{i+1} \in V_p^{n_i}$. Independently of the state of these two nodes, it must be true that:

$$\begin{array}{c}(fd_p^{n_i}(t) > fd_p^{n_{i+1}}(t)) \vee \\ (fd_p^{n_i}(t) = fd_p^{n_{i+1}}(t) \wedge |n_i| > |n_{i+1}|)\end{array} \quad (9)$$

Proof: Consider the case that router n_i is *PASSIVE*, then from Lemma 3 and the fact that routers select their next hops based on *NSC*, it must be true that:

$$\begin{array}{c}(fd_p^{n_i} > d_{pn_{i+1}}^{n_i}(t)) \vee \\ (fd_p^{n_i} = d_{pn_{i+1}}^{n_i}(t) \wedge |n_i| > |n_{i+1}|)\end{array} \quad (10)$$

Consider the case that router n_{i+1} is *ACTIVE*. Router n_{i+1} cannot change its successor or increase its feasible distance. If router n_i processed the last message that router n_{i+1} sent before time t, then: $d_{pn_{i+1}}^{n_i}(t) = fd_p^{n_{i+1}}(t)$ and the lemma is true. Assume router n_i did not process the last message that router n_{i+1} sent before time t. Router n_i must send a *REPLY* to n_{i+1} the last time that router n_{i+1} became *PASSIVE* at time t_p reporting a distance $d_p^{n_{i+1}}(t_{old}) = d_{ps_{old}}^{n_{i+1}} + l_{s_{old}}^{n_{i+1}}$.

If router n_{i+1} did not reset its feasible distance since the last time it became passive, $fd^{n_{i+1}}$, then, $d_p^{n_{i+1}}(t_{old}) \geq fd_p^{n_{i+1}}(t)$. Consider the case that router n_{i+1} resets $fd^{n_{i+1}}$ the last time before t that it becomes *PASSIVE*. Router n_{i+1} cannot change its successor or experience any increment in its distance through its old successor, s_{old}. Hence, $d_p^{n_{i+1}}(t_{new}) \leq d_p^{n_{i+1}}(t_{old})$. On the other hand, the distance through the new successor must be the smallest among all neighbors including the old successor and so $d_p^{n_{i+1}}(t_{new}) = (d_{ps_{new}}^{n_{i+1}} + l_{s_{new}}^{n_{i+1}}) \leq d_p^{n_{i+1}}(t_{old})$. Router n_{i+1} becomes *PASSIVE* if it receives a *REPLY* from each of its neighbors, including n_i. Therefore, n_i must be notified about $d_p^{n_{i+1}}(t_{old})$. Therefore,

$$d_{pn_{i+1}}^{n_i}(t) = d_p^{n_{i+1}}(t_{old}) \geq d_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_{i+1}}(t_{new}) \quad (11)$$

The feasible distance $fd_p^{n_{i+1}}(t_{new})$ with $t_{new} < t$ cannot increase until router n_{i+1} becomes *PASSIVE* again; therefore, $fd_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_i}(t)$. The result of the lemma follows in this case by substituting this result in Eqs. (11) and Eq. (10).

Now consider the case that router n_{i+1} is *PASSIVE*. If router n_i processed the last message that router n^{i+1} sent before time t, then $d_{pn_{i+1}}^{n_i}(t) = d_p^{n_{i+1}}(t) \geq fd_p^{n_{i+1}}(t)$ and the lemma is true. Now consider the case that router n_i did not process the last message router n^{i+1} sent before time t. If router n_{i+1} did not reset $fd^{n_{i+1}}$ then $d_p^{n_{i+1}}(t_{old}) \geq fd_p^{n_{i+1}}(t)$. On the other hand, if router n_{i+1} resets $fd^{n_{i+1}}$ then we can conclude that $fd_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_i}(t)$ and $|n_i| > |n_{i+1}|$ using an argument similar to one we used for the *ACTIVE* mode. Hence, the lemma is true for all cases. ■

NSC and *SRC* guarantees loop-freedom at every time instant. If we consider the link form router i to its valid next hop with respect to a specific prefix as a directed edge, then the graph containing all this directed links is a directed acyclic graph (DAG) with respect to that specific prefix. The DAG representing the relationship of valid next hops regarding prefix p is denoted by D_p.

Lemma 5: If routers are involved in a single diffusing computation then D_p is loop-free at every instant.

Proof: Assume for the sake of contradiction that D_p is loop-free before an arbitrary time t and a loop L_p consisting of h hops is created at time $t_l > t$ when router q updates V_p^q after

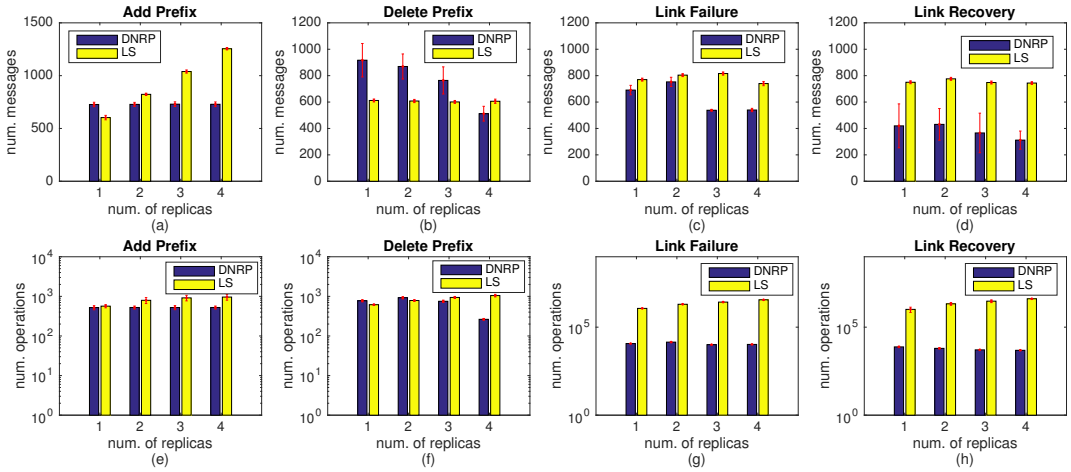


Fig. 2. Simulation results showing average number of messages and average number of operations vs number of replicas

processing an input event. Assume that $L_p = \{n_1, n_2, \ldots, n_h\}$ is the loop created, where $n_{i+1} \in V_p^{n_i}$ for $1 \leq i \leq h$ and $n_1 \in V_p^{n_h}$. If router n_1 changes its next hop as a result of changing its successor, it must be in *PASSIVE* mode at time t_l because an *ACTIVE* router cannot change its successor or update its next-hop set.

If all routers in L_p are *PASSIVE* at time t_l, either all of them have always been *PASSIVE* at every instant before t_l, or at least one of them was *ACTIVE* for a while and became *PASSIVE* before t_l. If no router was ever *ACTIVE* before time t_l, it follows from Theorem 1 that updating V_p^n cannot create loop. Therefore, for router n_1 to create a loop, at least one of the routers must have been *ACTIVE* before time t.

If all routers are in *PASSIVE* mode at time t, traversing L_p and applying Theorem 10 leads to the erroneous conclusion that either $d_p^{n_1} > d_p^{n_1}$ or $|n_1| > |n_1|$. Therefore updating $V_p^{n_1}$ cannot create a loop if all routers in the L_p are *PASSIVE* at time t.

Assume that only one diffusing computation is taking place at time t_l. Based on Lemma 4 traversing loop L_p leads to the conclusion that either $fd^{n_i} > fd^{n_i}$ or $|n_i| > |n_i|$, which is a contradiction. Therefore, if only a single diffusing computation takes place, then L_p cannot be formed when routers use *SRC* and *NSC* along with difusing computations to select next hops to reach the destination prefix. ■

At steady state, the graph containing the successors and connected links between them, must create a tree. The tree containing successors that are *ACTIVE* regarding prefix p and participating in a diffusion computation started form router i at time t are called diffusing tree ($T_{pi}(t)$).

Theorem 6: DNRP considers each computation individually and in the proper sequence.

Proof: Assume router i is the only router that has started a diffusing computation up to time t. If router i generates a single diffusion computation, the proof is immediate. Consider the case that router i generates multiple diffusing computations. Any router that is already participating in the current diffusing computation (routers in the T_{pi}, including the router i) cannot send a new *QUERY* until it receives all the replies to the *QUERY* of the current computation and becomes *PASSIVE*. Note that each router processes each event in order. Also, when a router becomes *PASSIVE*, it must send a *REPLY* to its successor, if it has any. Therefore, all the routers in T_{pi} must process each diffusing computation individually and in the proper sequence.

Consider the case that multiple sources of diffusing computations exist regarding prefix p in the network. Assume router i is *ACTIVE* at time t. Then either router i is the originator of the diffusing computation ($o_p^i = 1$ or 2), or received a *QUERY* from its successor ($o_p^i = 3$ or 4). If $o_p^i = 1$ or 3, the router must become *PASSIVE* before it can send another *QUERY*. If the router is the originator of the computation ($o_p^i = 1$ or 2) and receives a *QUERY* form its successor, it holds that *QUERY* and sets o_p^i =4. Therefore, all the routers in the T_{pi} remain in the same computation. Router i can forward the new *QUERY* and become the part of the larger $T_p s$ only after it receives a *REPLY* form each of its neighbors for the current diffusing computation. If router a is *ACTIVE* and receives a *QUERY* from its neighbor $k \neq s_p^a$, then it sends a *REPLY* to its neighbor before creating a diffusing computation, which means that T_{pa} is not part of the *ACTIVE* T_p to which k belongs. Therefore, any two *ACTIVE* T_{pi} and T_{pj} have an empty intersection at any given time, it thus follows from the previous case that the Theorem is true. ■

V. Performance Comparison

We compare DNRP with a link-state routing protocol given that NLSR [13] is based on link states and is the routing protocol advocated in NDN, one of the leading ICN architectures. We implemented DNRP and an idealized version of NLSR, which we simply call ILS (for ideal link-state), in ns-3 using the needed extensions to support content-centric networking [19]. In the simulations, ILS propagates update messages using the intelligent flooding mechanism. There are two types of Link State Advertisements (LSA): An adjacency

LSA carries information regarding a router, its neighbors, and connected links; and a prefix LSA advertises name prefixes, as specified in [13]. For convenience, DNRP sends HELLO messages between neighbors to detect changes in the sate of nodes and links. However, HELLO's can be omitted in a real implementation and detecting node adjacencies can be done my monitoring packet forwarding success in the data plane.

The AT&T topology [20] is used because it is a realistic topology for simulations that mimic part of the Internet topology. It has 154 nodes and 184 links. A node has 2.4 neighbors on average. In the simulations, the cost of a link is set to one unit, and 30 nodes are selected as anchors that advertise 1200 unique name prefixes. We generated test cases consisting of single link failure and recovery, and a single prefix addition and deletion.

To compare the computation and communication overhead of DNRP and ILS, we measured the number of routing messages transmitted over the network and the number of operations executed by each routing protocol. The number of messages for ILS includes the number of HELLO messages, Adjacency LSAs, and Prefix LSAs. For DNRP, this measurement indicates the total number of all the routing messages transmitted as a result of any changes. The operation count is incremented whenever an event occurs, and statements within a loop are executed.

The simulation results comparing DNRP with ILS are depicted in Figure 2. In each graph, the horizontal axes is the average number of anchors per prefix, i.e., the number of anchors that advertise the same prefix to the network. We considered four scenarios: adding a new prefix to the network; deleting one prefix from one of the replicas; a single link failure; and a single link recovery. Hence, ILS incurs the same signaling overhead independently of how many LSA's are carried in an update. Figures (2a - 2d) showthe number of messages transmitted in the whole network while Figures (2e - 2h) show the number of operations each protocol executed after the change. The number of operations in the figure is in logarithmic scale.

ILS advertises prefixes from each of the replicas to the whole network. As the number of replicas increases, the number of messages increases, because each replica advertises its own Prefix LSA. In DNRP, adding a new prefix affects nodes in small regions and hence the number of messages and operations are fewer than in ILS. Deleting a prefix from one of the replicas results in several diffusing computations in DNRP, which results in more signaling. However, the number of messages decreases as the number of replicas increases, because the event affect fewer routers. In ILS one Prefix LSA will be advertised for each deletion. The computation of prefix deletion is comparable; however, DNRP imposes less computation overhead when the number of replicas reach 4.

DNRP has less communication overhead compared to ILS after a link recovery or a link failure. The need to execute Dijkstra's shortest-path first for each neighbor results in ILS requiring more computations than DNRP. DNRP outperforms NLSR for topology changes as well as adding a new prefix.

VI. Conclusion

We introduced the first name-based content routing protocol based on diffusing computations (DNRP) and proved that it provides loop-free multi-path routes to multi-homed name prefixes at every instant. Routers that run DNRP do not require to have knowledge about the network topology, use complete paths to content replicas, know about all the sites storing replicas of named content, or use periodic updates. DNRP has better performance compared to link-state routing protocols when topology changes occur or new prefixes are introduced to the network. A real implementation of DNRP would not require the use of HELLO's used in our simulations, and hence its overhead is far less than routing protocols that rely on LSA's validated by sequence numbers, which require periodic updates to work correctly.

VII. Acknowledgments

This work was supported in part by the Baskin Chair of Computer Engineering at UCSC.

References

[1] M. Bari et al., "A Survey of Naming and Routing in Information-Centric Networks," *IEEE Communications Magazine*, vol. 50, no. 12, pp. 44–53, Dec. 2012.

[2] J. Behrens and J.J. Garcia-Luna-Aceves, "Hierarchical Routing Using Link Vectors," *Proc. IEEE INFOCOM '98*, March 1998.

[3] J. Choi et al., "A Survey on Content-Oriented Networking for Efficient Content Delivery," *IEEE Communications Magazine*, March 2011.

[4] J. J. Garcia-Luna-Aceves, "A Distributed, Loop-Free, Shortest-Path Routing Algorithm," *Proc. IEEE INFOCOM '88*, Mar 1988.

[5] J. J. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations," *IEEE/ACM Transactions on Networking*, 1993.

[6] J.J. Garcia-Luna-Aceves and M. Spohn, "Scalable Link-State Internet Routing," *Proc. ICNP '98*, Oct. 1998.

[7] J.J. Garcia-Luna-Aceves, "System and Method for Discovering Information Objects and Information Object Repositories in Computer Networks," U.S. Patent 8,572,214, October 29, 2013.

[8] J. J. Garcia-Luna-Aceves, "Name-Based Content Routing in Information Centric Networks Using Distance Information," in *Proc. ACM ICN '14*, 2014.

[9] J. J. Garcia-Luna-Aceves, "Routing to Multi-Instantiated Destinations: Principles and Applications," *Proc. IEEE ICNP 2014*, 2014.

[10] J. J. Garcia-Luna-Aceves, "A Fault-Tolerant Forwarding Strategy for Interest-Based Information Centric Networks," *Proc. IFIP Networking '15*, 2015.

[11] K. V. Katsaros et al., "On Inter-Domain Name Resolution for Information-Centric Networks," *Proc. Networking 2012*, May 2012.

[12] T. Koponen et al., "A Data-Oriented (and Beyond) Network Architecture," *Proc. ACM SIGCOMM '07*, 2007.

[13] V. Lehman et al., "A Secure Link State Routing Protocol for NDN," *IEEE Access*, Jan. 2018

[14] Mobility first project. [Online]. Available: http://mobilityfirst.winlab.rutgers.edu/

[15] Publish subscribe internet technology (PURSUIT) project. [Online]. Available: http://www.fp7-pursuit.eu/PursuitWeb/

[16] J. Raju et al., "System and Method for Information Object Routing in Computer Networks," U.S. Patent 7,552,233, June 23, 2009

[17] M. Spohn and J.J. Garcia-Luna-Aceves, "Neighborhood Aware Source Routing," *Proc. ACM MobiHoc 2001*, Oct. 2001.

[18] Scalable and adaptive internet solutions (SAIL) project. [Online]. Available: http://www.sail-project.eu/

[19] J. Mathewson et al., "Sconet : Simulator content networking," *CCNxCon*, 2015.

[20] O. Heckmann et al., "On realistic network topologies for simulation" *MoMeTools '03* , 2003.

PopNetCod: A Popularity-based Caching Policy for Network Coding enabled Named Data Networking

Jonnahtan Saltarin*, Torsten Braun*, Eirina Bourtsoulatze† and Nikolaos Thomos‡
*University of Bern, Bern, Switzerland
†Imperial College London, London, United Kingdom
‡University of Essex, Colchester, United Kingdom
saltarinj@gmail.com, braun@inf.unibe.ch, e.bourtsoulatze@imperial.ac.uk, nthomos@essex.ac.uk

***Abstract*—In this paper, we propose PopNetCod, a popularity-based caching policy for network coding enabled Named Data Networking. PopNetCod is a distributed caching policy, in which each router measures the local popularity of the content objects by analyzing the requests that it receives. It then uses this information to decide which Data packets to cache or evict from its content store. Since network coding is used, partial caching of content objects is supported, which facilitates the management of the content store. The routers decide the Data packets that they cache or evict in an online manner when they receive requests for Data packets. This allows the most popular Data packets to be cached closer to the network edges. The evaluation of PopNetCod shows an improved cache-hit rate compared to the widely used Leave Copy Everywhere placement policy and the Least Recently Used eviction policy. The improved cache-hit rate helps the clients to achieve higher goodput, while it also reduces the load on the source servers.**

I. Introduction

Data intensive applications, *e.g.*, video streaming, software updates, *etc.*, are the major sources of data traffic in the Internet, and their predominance is expected to further increase in the near future [1]. Moreover, nowadays Internet users are more concerned about *what* data they request, rather than *where* that data is located. To address the increased data traffic and the shift in interest from location to data, technologies like Content Delivery Networks (CDN) have been proposed. However, these solutions cannot fully exploit the network resources and deal effectively with the increasing amount of data traffic, since they work on top of the current Internet architecture, which is based on host-to-host communication. To address this issue, the Named Data Networking (NDN) architecture [2], [3] has been proposed, which replaces the addresses of the communicating hosts (*i.e.*, IP addresses) with the name of the data being communicated. In the NDN architecture, clients request data by sending an *Interest* that contains the name of the requested data. Any network node that receives the Interest and holds a copy of the requested data can satisfy it by sending a *Data packet* back to the client.

Two of the main advantages that the NDN architecture has over the traditional host-to-host architectures are: *(i)* the inherent use of in-network caching, and *(ii)* the built-in support for multipath communications. The pervasive in-network caching concept proposed by NDN reduces the number of hops that Interests and Data packets need to travel in the network. This reduces the delay perceived by the application retrieving the requested data. However, having caches in all the routers is not always necessary to yield the full benefits that caching brings to the data delivery process. Previous works [4]–[6] have shown that enabling caches only at the edge of the network may achieve performance improvements similar to those obtained when every router is equipped with a cache. Furthermore, NDN provides natural multipath support by allowing clients to distribute the Interests that they need to send to retrieve content objects over all their network interfaces (*e.g.*, LTE, Wi-Fi), which enables the applications to better use the clients' network resources. However, in the presence of multiple clients and/or multiple data sources, the optimal use of multiple paths requires the nodes to coordinate where they forward each Interest in order to reduce the number of Data packet transmissions and the network load.

To optimally exploit the benefits brought by in-network caching and multipath communication, previous works [7], [8] had proposed the use of network coding [9]. In a network coding enabled NDN architecture, the network routers code Data packets by combining the Data packets available at their caches prior to forwarding them. The use of network coding *(i)* increases Data packet diversity in the network, hence, the use of in-network caches is optimized, and *(ii)* in multi-client and multi-source scenarios it removes the need for coordinating the faces where the nodes forward each Interest, which enables efficient multipath communication. Although there are works that consider the use of network coding in NDN, they do not consider that caching capacity is limited [7], [8], [10], [11] or they assume that a centralized node coordinates the caching decisions [12], [13], which is unrealistic or difficult to deploy.

In this paper, our goal is to develop a distributed caching policy that preserves the benefits that network coding brings to NDN for the realistic case when the caches have limited capacity. We propose *PopNetCod*, a popularity-based caching policy for network coding enabled NDN architectures. PopNetCod is a caching policy in which routers distributedly estimate the popularity of the content objects based on the received Interest. Based on this information, each router decides which Data packets to insert or evict from its cache. The decision to cache a particular Data packet is taken before the Data packet arrives at the router, *i.e.*, while processing the corresponding

ISBN 978-3-903176-08-9

Interest. Since the first routers to process Interests in their path to the source are the edge routers, this helps to cache the most popular Data packets closer to the network edges, which reduces the data delivery delay [4]–[6]. To avoid caching the same Data packet in multiple routers over the same path, routers communicate the Data packets that they decide to cache by setting a binary flag in the Interests to be forwarded upstream. This increases the Data packet diversity in the caches. When the cache of a router is full and a Data packet should be cached, the router decides which Data packet should be evicted from its cache based on the popularity information.

We implement the proposed caching policy on top of ndnSIM [14], based on the NetCodNDN codebase [8], [10]. We evaluate the performance of PopNetCod in a Netflix-like video streaming scenario, designed using parameters available in the literature [15]–[17]. In comparison with a caching policy that uses the NDN's default Leave Copy Everywhere (LCE) placement policy and the Least Recently Used (LRU) eviction policy, PopNetCod achieves a higher cache-hit rate, which translates into higher video quality at the clients and reduced load at the sources.

The remainder of this paper is organized as follows. Section II provides an overview of the related works. Section III describes the system architecture. Section IV introduces the problem of caching in network coding enabled NDN for data intensive applications. Then, Section V presents our caching policy, PopNetCod. A practical implementation of the PopNetCod caching policy is described in Section VI. Section VII presents the evaluation of the PopNetCod caching policy.

II. Related Work

Caching policies are needed to deal with caches that have limited capacity. Caching policies decide which Data packets are placed into the cache (*placement*), as well as which data packets are evicted from the cache when the cache is full and a new Data packet should be cached (*eviction*). There are placement algorithms that consider content popularity to decide which Data packets routers allow in their caches [18]–[21]. Specifically, *VIP* [18] is a framework for joint Interest forwarding and Data packet caching. This scheme uses a "virtual control plane" that operates on the Interest rate and a "real plane" which handles Interests and Data packets. It is shown that the design of joint algorithms for routing and caching is important for NDN. Thus, this scheme proposes distributed control algorithms that operate in the virtual control plane with the aim of increasing the number of Interests satisfied by in-network caches. *PopCaching* [19] is a popularity-based caching policy in which the popularity is computed online, without the need for a training phase. This makes *PopCaching* robust in dynamic popularity settings. However, *PopCaching* is designed for caching systems with a single cache in the path, while in this paper we are interested in networks of caches. *WAVE* [20] is a placement algorithm that determines the number of Data packets that should be cached for a given file with the help of an access counter. The number of Data packets to cache increases exponentially with the value of the access counter. The main idea of *WAVE*, which partially caches a content object according to the local popularity, is also adopted by the caching policy that we propose in this paper. However, *WAVE* does not facilitate edge caching, since the most popular data is cached closer to the source and slowly moves towards the edges as the number of requests increase. *Progressive* [21] is another partial caching algorithm, which exploits the content popularity to decide how many Data packets should be cached for each name prefix. The cache placement decision is taken when the Interests are received, which helps to cache the most popular content at the network edge. However, this approach lacks an eviction algorithm, and hence it cannot be deployed when the cache capacity is limited.

None of the approaches above consider the use of network coding [9], and all are evaluated in single-path scenarios. Given the benefits that network coding brings to multipath communications in NDN [7], [8], [10], [11], some approaches have been proposed to improve the benefits of caching in network coding enabled NDN architectures [12], [13], [22]. *NCCAM* [12] and *NCCM* [13] propose optimal solutions to the problem of efficiently caching in network coding enabled NDN. However, both approaches need a central entity that is aware of the network topology and the Interests, which does not scale well with the number of network nodes. *CodingCache* [22] is an eviction policy in which routers, before evicting a Data packet, apply network coding to the Data packet by means of combining it with other Data packets with the same name prefix that will remain in the cache. Due to the increased Data packet diversity in the network, the cache-hit rate is improved. However, in *CodingCache* Interest aggregation and Interest pipelining are problematic, limiting the benefits that network coding brings to the NDN architecture.

III. Overview of Network Coding Enabled NDN

A. Data Model

We consider a set of content objects $\mathcal{P}$ that is made available by a *content provider* to a set of *end users*. Each content object is uniquely identified by a name n. Clients use this name to request that particular content object. Each content object is divided into a set of Data packets $\mathcal{P}_n$, such that the size of each Data packet does not exceed the Maximum Transmission Unit (MTU) of the network. The set of Data packets $\mathcal{P}_n$ that compose a content object is divided into smaller sets of Data packets, which are known as *generations* [23]. The size of each generation g is a design parameter chosen to enable network coding at scale. The set of Data packets that form the generation g is denoted as $\hat{\mathcal{P}}_{n,g}$ and a network coded Data packet belonging to generation g is represented by $\hat{p}_{n,g}$.

B. Router Model

The routers have three main tables: a *Content Store (CS)*, where they cache Data packets to reply to future Interests, a *Pending Interest Table (PIT)*, where they keep track of the Interests that have been received and forwarded, to know where to send the Data packets backward to the clients, and a

Forwarding Information Base (FIB), which associates upstream faces with name prefixes, to route the Interests towards the sources. In order to enable the use of caching policies in the NetCodNDN architecture, we extend its design by adding a new module called Content Store Manager (CSM). The CSM manages the content store by enforcing a determined caching policy.

Whenever a router receives an Interest $\hat{i}_{n,g}$, it first verifies if it can reply to this Interest with the Data packets available in the CS. The router replies to the Interest if it is able to generate a network coded Data packet that has high probability of being innovative when forwarded on the path where the Interest arrived, *i.e.*, if the generated Data packet is linearly independent with respect to all the Data packets that have been sent over the face where the Interest arrived. In this case, the router generates a new Data packet by randomly combining the Data packets in its CS and then sends it downstream over the face where the Interest arrived. Otherwise, the router forwards the Interest to its upstream neighbors to receive a new Data packet that enables it to satisfy this Interest. However, if the router has already forwarded one or multiple Interests with the same name prefix (n, g) and it expects to receive enough Data packets to reply to all the pending Interests stored in the PIT, the router simply *aggregates* this Interest in the PIT, and waits for enough innovative Data packets to arrive before replying to the Interest.

Whenever a router receives a Data packet $\hat{p}_{n,g}$, it first determines if the Data packet is innovative or not. A Data packet $\hat{p}_{n,g}$ is innovative if it is linearly independent with respect to all the Data packets in the CS of the router, *i.e.*, if it increases the rank of $\hat{\mathbf{P}}^r_{n,g}$. Non-innovative Data packets are discarded. If the Data packet $\hat{p}_{n,g}$ is innovative, the router sends the Data packet to the CSM, which decides to cache it or not according to the caching policy. Finally, the router generates a new network coded Data packet and sends it over every face that has a pending Interest to be satisfied.

C. Content Store Model

The Content Store (CS) is a temporary storage space in which a router r can cache Data packets that it has received and considers useful to reply to future Interests. The maximum number of Data packets that can be cached in the CS is given by M, while the set of Data packets that are cached in the CS is denoted as $\hat{\mathcal{P}}^r$. Thus, $|\hat{\mathcal{P}}^r| \leq M$.

Data packets in the CS are organized in CS entries. Each CS entry contains a set of network coded Data packets, $\hat{\mathcal{P}}^r_{n,g}$, that belong to the same generation g. Since the CS has a limited capacity of M Data packets, then $\sum_{n,g} |\hat{\mathcal{P}}^r_{n,g}| \leq M$. The Data packets that compose a CS entry are stored in a matrix $\hat{\mathbf{P}}^r_{n,g}$, where each row is a vector $\hat{\mathbf{p}}_{n,g}$ that represents the network coded Data packet $\hat{p}_{n,g}$.

Router r generates a network coded Data packet $\hat{p}_{n,g}$ by randomly combining the Data packets $\hat{\mathbf{P}}^r_{n,g}$ in its CS. Thus, $\hat{p}_{n,g} = \sum_{j=1}^{|\hat{\mathbf{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$, where a_j is a randomly selected coding coefficient and $\hat{\mathbf{p}}^{(j)}_{n,g}$ is the jth Data packet in $\hat{\mathbf{P}}^r_{n,g}$.

Additionally to the matrix $\hat{\mathbf{P}}^r_{n,g}$, each CS entry also stores a counter $\sigma^f_{n,g}$ for each face f of router r. This counter measures the number of Data packets generated by applying network coding to the Data packets stored in matrix $\hat{\mathbf{P}}^r_{n,g}$ that have already been sent over face f, *i.e.*, it measures the amount of information from matrix $\hat{\mathbf{P}}^r_{n,g}$ that has been transmitted from router r to its neighboring node connected over face f. The counter $\sigma^f_{n,g}$ is used to compute the number of network coded Data packets with name prefix (n, g) that the router can generate with the Data packets cached in its CS and have high probability of being innovative to its neighboring node connected over face f. This number is denoted as $\xi^f_{n,g}$ and is computed as follows:

$$\xi^f_{n,g} = \texttt{rank}(\hat{\mathbf{P}}^r_{n,g}) - \sigma^f_{n,g}. \tag{1}$$

When a Data packet with name prefix (n, g) is evicted from the CS of router r, the amount of information in the matrix $\hat{\mathbf{P}}^r_{n,g}$ is reduced by 1. Correspondingly, the value of $\sigma^f_{n,g}$ is decreased by 1 for all faces.

IV. Caching in Network Coding Enabled NDN

Whenever a router r receives an Interest $\hat{i}_{n,g}$ over face f, it either *(i)* replies with a Data packet $\hat{p}_{n,g}$, if it can generate a network coded Data packet that has high probability of being innovative to its neighboring node connected over face f, *i.e.*, $\xi^f_{n,g} > 0$, or, otherwise, *(ii)* forwards the Interest $\hat{i}_{n,g}$ upstream.

If at time t router r receives the Interest $\hat{i}_{n,g}$, a cache-hit is defined as:

$$h^f_{n,g}(t) = \begin{cases} 1, & \text{if } \xi^f_{n,g} > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Let us now assume that during a time period $[t, t+T]$ router r receives a set of Interests $\mathcal{I}(t, T)$. The cache-hit rate during this time period is defined as follows:

$$H(t, T) = \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{3}$$

The overall cache-hit rate seen by router r at time t can be computed as follows:

$$H(t) = \lim_{T\to\infty} H(t, T) = \lim_{T\to\infty} \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{4}$$

To make optimal use of the limited CS capacity, the objective of each router is to maximize the number of Interests that it can satisfy with the Data packets available in its CS, *i.e.*, maximize its overall cache-hit rate. Achieving a high cache-hit rate at the routers is beneficial for both clients and sources. For the sources, an increased cache-hit rate reduces their processing load and bandwidth needs, since the number of Interests that they receive is reduced. For the clients, the delivery delay is reduced, since the Interests are satisfied with Data packets cached at routers closer to them.

It is clear from (2), (3), and (4) that in order to maximize the overall cache-hit rate, routers should maintain the value of $\xi^f_{n,g}$ high enough so that most of the Interests received can

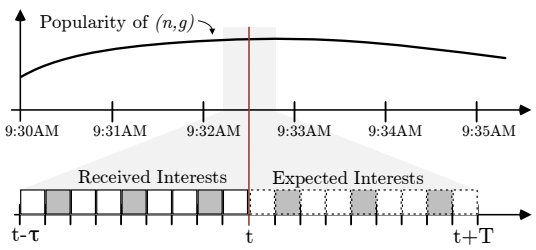


Fig. 1. Popularity prediction for the name prefix (n, g).

be satisfied with the Data packets in their CS. However, since in this paper we consider that the routers' CS have limited capacity, it is unfeasible for a router to cache all the Data packets that it receives [7], [8], [10], [11]. Optimal solutions to this issue have been proposed in previous works [12], [13], which consider a central controller that knows the network topology and is aware of all the Interests received by the routers. However, these solutions do not scale well with the size of the network, since they require a high number of signaling messages and a powerful enough controller. Hence, in this work we consider that each router decides online and independently from other routers if a Data packet should be cached or not, and which Data packet should be evicted from the CS when it is full. This is achieved by using a distributed caching policy π that maximizes the overall cache-hit rate $H(t)$ of each router,

$$\max_{\pi} H(t). \tag{5}$$

The optimal caching policy π predicts which Interests will be received in the future, so that the router caches the Data packets that will be useful to satisfy those Interests.

V. The PopNetCod Caching Policy

In this section, we present our popularity-based caching policy for network coding enabled NDN, called PopNetCod. To increase the overall cache-hit rate, the PopNetCod caching policy exploits real-time data popularity measurements to determine the number of Data packets that each router should cache for each name prefix. In order to determine which Data packets to cache in and/or evict from the CS, such that the overall cache-hit rate is maximized, PopNetCod performs the following steps. First, it measures the popularity of the different name prefixes contained in the Interests that pass through it. Then, it uses this popularity to predict the Interests that it will receive. Finally, it uses this prediction to determine in an online manner the Data packets that should be cached and the ones that should be evicted from the CS.

A. Popularity Prediction

The popularity prediction in PopNetCod is based on the fact that the rate $\lambda^f_{n,g}(t)$ at which Interests for a particular content object arrive at a router r over face f at time t tends to vary smoothly, as shown in Fig. 1. Thus, router r can predict the rate of the Interests that it will receive in the near future by observing the Interests that it recently received. Let us denote $\mathcal{I}^f_{n,g}(\tau, t)$ as the set of Interests for the name prefix (n, g) that router r has received over face f in the past period $[t - \tau, t]$, where t is the current time and τ is the observation period. Let us also denote $\mathcal{I}^f(\tau, t)$ as the total set of Interests for all name prefixes received over face f during the period $[t - \tau, t]$. Using the sets $\mathcal{I}^f_{n,g}(\tau, t)$ and $\mathcal{I}^f(\tau, t)$, router r can compute the average Interest rate for the name prefix (n, g) over face f as follows:

$$\lambda^f_{n,g}(\tau, t) = \frac{|\mathcal{I}^f_{n,g}(\tau, t)|}{|\mathcal{I}^f(\tau, t)|}, \tag{6}$$

Note that since the average Interest rate does not vary abruptly, the average Interest rate $\lambda^f_{n,g}(\tau, t)$ of the recent period $[t - \tau, t]$ will be very close to that expected in the near future, *i.e.*, in the period $[t, t + T]$ where T is the length of the prediction period. Thus, $\lambda^f_{n,g}(\tau, t) = \lambda^f_{n,g}(t, T)$, which hereafter we denote as $\lambda^f_{n,g}(t)$. The PopNetCod caching policy uses $\lambda^f_{n,g}(t)$ to predict the number of Interests with name prefix (n, g) that will be received over face f in the near future, and hence, to allocate more storage space in the CS to Data packets with higher cache-hit probability.

In order to prepare the CS for the Interests that the router may receive, the PopNetCod caching policy maps the received Interest rate to the capacity of the CS, such that name prefixes with high rate are allocated more space in the CS. The number of network coded Data packets with name prefix (n, g) that the router should cache in its CS at time t to satisfy the Interests expected over face f is denoted as $M^f_{n,g}(t)$ and computed as:

$$M^f_{n,g}(t) = \begin{cases} \lambda^f_{n,g}(t) \cdot M, & \text{if } \lambda^f_{n,g}(t) \cdot M < |\hat{\mathcal{P}}_{n,g}| \\ |\hat{\mathcal{P}}_{n,g}|, & \text{otherwise.} \end{cases} \tag{7}$$

B. PopNetCod Placement

In the PopNetCod caching policy, the placement decision is taken following the reception of an Interest. Whenever a router decides to cache the Data packet that is expected as a reply to the received Interest, it sets a flag on the Interest signaling upstream routers about its decision. In the case of a set flag, the upstream nodes do not consider this Interest for caching. Since the edge routers (*i.e.*, the routers that are directly connected to the clients) are the first ones that have the possibility to decide whether they will cache a Data packet, the PopNetCod caching policy naturally enables edge caching. This is inline with recent works [4]–[6] arguing that most of the gains from caching in NDN networks come from edge caches, and thus, it is natural to cache the most popular content at edge routers.

Whenever a router receives an Interest $\hat{i}_{n,g}$ over face f_t at time t, the PopNetCod caching policy follows the next steps to decide if the Data packet $\hat{p}_{n,g}$ should be cached. First, it uses popularity prediction to compute $M^f_{n,g}(t)$, *i.e.*, the total number of Data packets that it aims to cache for name prefix (n, g), as defined in (7). Then, it computes the number of Data packets that it should cache in order to satisfy the expected Interests as:

$$\delta^f_{n,g}(t) = M^f_{n,g}(t) - \xi^f_{n,g}(t) \ \forall f \in \mathcal{F}. \tag{8}$$

Finally, the caching policy decides to cache the Data packet $\hat{p}_{n,g}$ that is expected as reply to the received Interest if the average number of Data packets needed by all the faces is greater than 0. However, it should be noted that the Data packet $\hat{p}_{n,g}$ will not be useful to the node connected over the downstream face f_t over which the Interest arrived. This is because when the Data packet $\hat{p}_{n,g}$ arrives at the router, it is sent to face f_t in order to satisfy the received Interest. Then, replying with the same Data packet to a subsequent Interest received over the same face f_t does not add any innovative information, *i.e.*, the Data packet is considered as duplicated. Instead, the expected Data packet $\hat{p}_{n,g}$ is potentially useful for all the nodes connected over all the other downstream faces of the router. For this reason, the average number of Data packets needed is measured only over the downstream faces different to the one over which the Interest arrived. It is computed as:

$$\Delta^{+}_{n,g}(t) = \frac{1}{|\mathcal{F}^r| - 1} \sum_{\substack{f \in \mathcal{F} \\ f \neq f_t}} \delta^{f}_{n,g}(t) > 0, \tag{9}$$

where $\mathcal{F}^r$ denotes the downstream faces of router r.

C. PopNetCod Eviction

The steps followed by the PopNetCod caching policy to decide how many Data packets with name prefix (n, g) can be evicted from the router's CS are the following. Similarly to the placement case, first, the caching policy uses popularity prediction to compute $M^{f}_{n,g}(t)$, *i.e.*, the number of Data packets that it aims to cache for name prefix (n, g). Then, it computes the number of Data packets that it can evict from its CS and still satisfy the expected Interests as:

$$\tilde{\delta}^{f}_{n,g}(t) = \texttt{rank}(\hat{\mathbf{P}}^{r}_{n,g}) - M^{f}_{n,g}(t) \forall f \in \mathcal{F}. \tag{10}$$

Finally, the number of Data packets the router can evict from a particular name prefix (n, g) is computed as the minimum number of Data packets that it can evict over all the faces:

$$\Delta^{-}_{n,g}(t) = \min_{f \in \mathcal{F}} \tilde{\delta}^{f}_{n,g}(t). \tag{11}$$

VI. Practical Implementation of PopNetCod

In this section, we describe a practical implementation of the PopNetCod caching policy in the NetCodNDN architecture [10]. First, we describe the signaling between routers, which is used to prevent routers of the same path to cache duplicate Data packets. Next, we present the Interest processing algorithm, where placement decisions are made. Finally, we describe the Data packet processing algorithm for placement enforcement, eviction decision, and eviction enforcement.

A. Signaling Between Routers

The PopNetCod caching policy is distributed and requires very limited signaling between routers. The only signaling that exists between routers to implement the PopNetCod caching policy is a binary flag added to the Interest and Data packets that is used to inform neighbor routers that an expected Data packet will be cached or that a received Data packet has been cached. Distributed caching policy decisions help to keep the complexity of the system low and to make our system scalable to a large number of routers.

Each Interest $\hat{i}_{n,g}$ carries a flag `CachingDown`, which is set to 1 by a router when it decides to cache the Data packet $\hat{p}_{n,g}$ that is expected to come as reply to the Interest. This flag informs upstream routers that another router downstream has already decided to cache the Data packet that is expected to come as reply to this Interest. The routers receiving an Interest with the `CachingDown` flag set to 1 do not consider to cache the Data packet that is expected to come as reply to this Interest, therefore reducing the number of duplicated Data packets in the path and the processing load in the nodes.

Since Interests for network coded data do not request particular Data packets, but rather any network coded Data packet with the requested name prefix, the routers need a way to know that a Data packet has been already cached by another router, so that they avoid caching duplicated Data packets. For this reason, each Data packet $\hat{p}_{n,g}$ has a flag `CachedUp`, which is set to 1 by a router when it caches this Data packet in its CS. This flag informs the downstream routers that another router has already cached this Data packet. A router receiving a network coded Data packet with the flag set to 1 does not consider it for caching. Instead, it waits for another Data packet with the same name prefix that has not been cached upstream. This ensures that a Data packet is cached by only one router on its way to the client.

B. Status Information at Routers

Each router implementing the PopNetCod caching policy should store information that assists to identify the Data packets that should be cached or evicted. In particular, the router needs to keep the *Recently received Interests* information to compute the popularity prediction. Moreover, since the placement decision takes place when the Interest is received, the router needs to remember the *Names to be cached*, such that the selected Data packets are cached when they arrive. Finally, since the popularity information can vary over time, the routers should keep a list with the *Names to consider for eviction*, which is used when they decide about eviction. Below, we describe the data structures used to store this information.

- *Recently received Interests* — The router maintains a list $\mathbf{L}^f$ for each face f of the router, where it stores the names of the Interests $\mathcal{I}^f(\tau, t)$ received over face f during the period $[\tau, t]$. The parameter τ controls how much into the past is observed by the router to compute the popularity prediction. Together with the name prefix, each element in $\mathbf{L}^f$ also stores the time t_i at which the Interest was received, such that it can be removed from $\mathbf{L}^f$ at time $t_i + \tau$.
- *Names to be cached* — The router maintains a table $\mathbf{A}$, where it stores the name prefixes (*i.e.*, the content object name appended with the generation ID) and the number of the Data packets that should be cached. When the router receives an Interest $\hat{i}_{n,g}$ and the PopNetCod caching policy decides that the network coded Data packet that is expected as reply should be cached, the router adds its name prefix (n, g) to the list $\mathbf{A}$. Then, whenever a network coded Data packet arrives, the

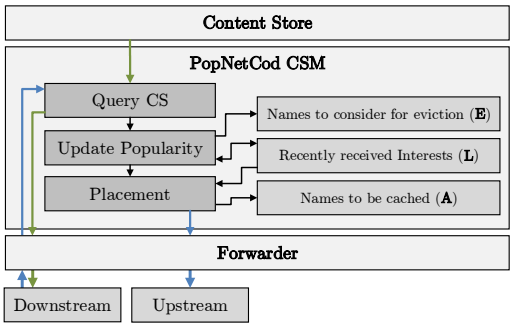


Fig. 2. Access to the CS and the Status Information during the Interest processing in a CSM configured with the PopNetCod caching policy.

Algorithm 1 Interest processing at the CSM

Require: $\hat{i}_{n,g}$, f

1: $t \leftarrow$ current time
2: **if** Flag `CachingDown` in $\hat{i}_{n,g}$ is set to 1 **then**
3: **if** $\xi^f_{n,g} > 0$ **then** ($\hat{i}_{n,g}$ *can be satisfied from the CS*)
4: Generate a Data packet $\hat{p}_{n,g}$ from the CS
5: Return $\hat{p}_{n,g}$
6: **else**
7: Return $\hat{i}_{n,g}$
8: **end if**
9: **else**
10: Add (n, g) to $\mathbf{L}^f$
11: **if** $\xi^f_{n,g} > 0$ **then** ($\hat{i}_{n,g}$ *can be satisfied from the CS*)
12: Generate a Data packet $\hat{p}_{n,g}$ from the CS
13: Return $\hat{p}_{n,g}$
14: **else if** $\hat{i}_{n,g}$ will be aggregated by the PIT **then**
15: Return $\hat{i}_{n,g}$
16: **else**
17: Update $\mathbf{L}$. (*Algorithm 2*)
18: **if** $\Delta^+_{n,g}(t) > 0$ **then** ($\hat{p}_{n,g}$ *should be cached*)
19: Insert (n, g) into $\mathbf{A}$
20: Set the flag `CachingDown` of $\hat{i}_{n,g}$ to 1
21: Return $\hat{i}_{n,g}$
22: **else**
23: Return $\hat{i}_{n,g}$
24: **end if**
25: **end if**
26: **end if**

router looks for the name prefix of the Data packet in the list $\mathbf{A}$. If it finds a match, it caches the Data packet.

- *Names to consider for eviction* — The router also maintains a queue $\mathbf{E}$, where it stores the name prefixes of the CS entries that can be considered for Data packet eviction. When a name prefix (n, g) is removed from the list $\mathbf{L}^f$, the popularity of this name prefix decreases, *i.e.*, it is a good candidate to consider for eviction. Thus, each time a name prefix is removed from $\mathbf{L}^f$, it is added to $\mathbf{E}$.

C. Interest Processing

As depicted in Fig. 2, when a CSM configured with the PopNetCod caching policy receives an Interest $\hat{i}_{n,g}$ from downstream, it *(i)* determines if the Interest can be replied from the CS. Then, if the CSM could not reply to the Interest with the content of its CS, it *(ii)* updates the popularity information, and, *(iii)* determines if the Data packet that is expected as reply to this Interest should be cached. The CSM should provide the NetCodNDN forwarder with either a Data packet that should be sent as reply to the Interest, or an Interest that should be forwarded upstream. Below we describe the details of this procedure, which is summarized in Algorithm 1.

Algorithm 2 Update $\mathbf{L}$

1: **for all** $f \in \mathcal{F}^r$ **do**
2: **for all** expired entries (n_l, g_l) in $\mathbf{L}^f$ **do**
3: Remove (n_l, g_l) from $\mathbf{L}^f$
4: Add (n_l, g_l) to $\mathbf{E}$
5: **end for**
6: **end for**

After receiving an Interest $\hat{i}_{n,g}$, the CSM first checks the flag `CachingDown` to see if any previous node downstream in the path has decided to cache the Data packet that is expected as reply to this Interest (lines 2 to 8). If the flag `CachingDown` is set to 1, then the CSM only checks its CS to determine if the Interest can be satisfied from the CS. If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder, which sends it over face f. If the Interest can not be satisfied from the CS, the CSM provides the same Interest to the NetCodNDN forwarder, which forwards it upstream.

If the flag `CachingDown` is set to 0, the CSM first inserts name (n, g) of the Interest into the list $\mathbf{L}^f$ (line 10). Then, the CSM checks if it can satisfy the Interest with the content of the CS (lines 11 to 13). If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder which sends it over face f. Otherwise, the node needs to forward the Interest to its neighbor nodes. If the router does not send the Interest upstream, but aggregates it in the PIT with a previously received Interest, the CSM does not need to do anything else and provides the Interest to the NetCodNDN forwarder, which aggregates it (line 15). If the Interest will not be aggregated, then the CSM determines if it will cache the Data packet with name prefix (n, g) that is expected as reply to this Interest, by computing $\Delta^-_{n,g}(t)$ using Eq. (9).

In order to obtain an accurate value of $\Delta^-_{n,g}(t)$, the CSM first updates the popularity information, removing all the expired elements from $\mathbf{L}^f$ and adding their name prefix to the list $\mathbf{E}$ of name prefixes to be considered for eviction (line 17). This procedure is summarized in Algorithm 2. Then, the CSM computes the value of $\Delta^+_{n,g}(t)$. If $\Delta^+_{n,g}(t) > 0$, it means that the Data packet should be cached. In this case, the CSM inserts name prefix (n, g) into the list $\mathbf{A}$, sets the flag `CachingDown` on the Interest $\hat{i}_{n,g}$ to 1 and, finally, provides the modified Interest to the NetCodNDN forwarder, which forwards it upstream (lines 18 to 21). If $\Delta^+_{n,g}(t) \leq 0$, then the CSM provides the same Interest to the NetCodNDN forwarder,

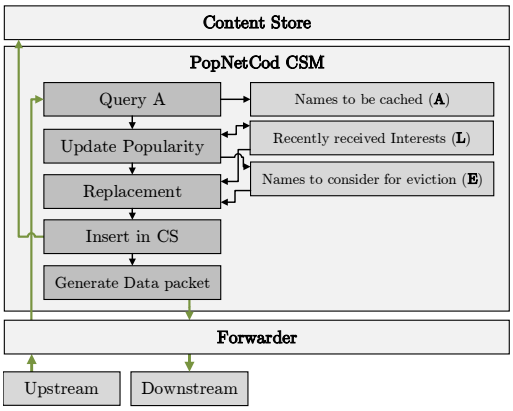


Fig. 3. Access to the CS and the Status Information during the Data packet processing in a CSM configured with the PopNetCod caching policy.

Algorithm 3 Data packet processing at the CSM

Require: $\hat{p}_{n,g}$
1: **if** Flag `CachingUp` in $\hat{p}_{n,g}$ is set to 1 **then**
2: Return $\hat{p}_{n,g}$
3: **else if** $(n,g) \notin \mathbf{A}$ **then**
4: Return $\hat{p}_{n,g}$
5: **else**
6: Update $\mathbf{A}$
7: **if** $|\mathcal{P}^r| == M$ **then** (*The CS is full*)
8: Update $\mathbf{L}$ (*Algorithm 2*)
9: **while** $|\mathcal{P}^r| == M$ **do**
10: Select an element (n_e, g_e) from $\mathbf{E}$
11: **if** $\Delta^-_{n_e,g_e}(t) > 0$ **then**
12: Evict $\Delta^-_{n_e,g_e}(t)$ Data packets with name prefix (n_e, g_e) from the CS
13: **end if**
14: **end while**
15: **end if**
16: Insert $\hat{p}_{n,g}$ into the CS
17: Generate a Data packet $\hat{p}^*_{n,g}$ from the CS
18: Set the flag `CachingDown` of $\hat{p}^*_{n,g}$ to 1
19: Return $\hat{p}^*_{n,g}$
20: **end if**

which forwards it upstream (line 23).

D. Data Packet Processing

As depicted in Fig. 3, when a CSM configured with the PopNetCod caching policy receives a network coded Data packet $\hat{p}_{n,g}$ from upstream, it *(i)* determines if the Data packet should be cached in the CS, by consulting $\mathbf{A}$. If the Data packet should be cached, the CSM ensures that there is enough free space in the CS, *(ii)* updating the popularity information and *(iii)* executing the cache replacement procedure if needed. Finally, the CSM *(iv)* inserts the received Data packet into the CS, and *(v)* generates a new network coded Data packet that should be forwarded downstream. This procedure is detailed below and summarized in Algorithm 3.

After receiving a Data packet $\hat{p}_{n,g}$, the CSM first checks the flag `CachedUp` to determine if any router upstream has already cached this Data packet. If the flag `CachedUp` has been set to 1, then, the CSM understands that another router upstream has already cached this Data packet. In this case, the CSM returns the Data packet to the NetCodNDN forwarder, which replies to any matching pending Interest (line 1).

When the flag `CachedUp` is set to 0, then the CSM first verifies if any entry in $\mathbf{A}$ matches name prefix (n,g). If there is no matching entry, the CSM returns the Data packet to the NetCodNDN forwarder (line 3). If there is a match, the Data packet should be cached, and $\mathbf{A}$ is updated by increasing the counter of the matching entry by one (line 6). However, if the CS is full, the CSM first needs to release some space in the CS (lines 7 to 15). To evict Data packets, the CSM goes through the list $\mathbf{E}$, each time selecting a name prefix (n_e, g_e) and computing the number of Data packets that can be evicted for the name prefix using Eq. (11). If this number is greater than 0, then the CSM evicts the corresponding number of Data packets from the CS and interrupts the scan of the list. Note that, since the cached Data packets are network coded, the CSM does not need to decide which particular Data packets from the CS entry $\hat{\mathcal{P}}_{n,g}$ it should evict from the CS, but it can select randomly network coded Data packets from the CS entry and evict them. After evicting at least one Data packet, the CSM caches the received Data packet $\hat{p}_{n,g}$. Then, the router generates a new Data packet $\hat{p}^*_{n,g}$ by applying network coding to the cached Data packets with name prefix (n,g). Since the new Data packet $\hat{p}^*_{n,g}$ contains the cached Data packet $\hat{p}_{n,g}$, the router sets the flag `CachedUp` of $\hat{p}^*_{n,g}$ to 1. Finally, the CSM provides Data packet $\hat{p}^*_{n,g}$ to the NetCodNDN forwarder, which uses it to reply to pending Interests with name prefix (n,g).

VII. Evaluation

In this section, we evaluate the performance of the PopNetCod caching policy in an adaptive video streaming architecture based on NetCodNDN [10]. First, we describe the evaluation setup. Then, we present the caching policies with which we compare the PopNetCod caching policy. Finally, we show the performance evaluation results.

A. Evaluation Setup

We consider a layered topology consisting of 1 source, 123 clients, and 45 routers connecting the clients and the sources. The routers are arranged in a two-tier topology, with 10 routers directly connected to the source and 35 edge routers directly connected to the clients. The links connecting the routers between them and the links connecting the routers to the source have a bandwidth of $20Mbps$. The bandwidth of the links connecting the clients to the routers follow a normal distribution, with mean $4Mbps$ and standard deviation 1.5. These values are chosen based on the Netflix ISP Speed Index [17]. Each client is connected with two routers, considering that nowadays most end-user devices have multiple interfaces, *e.g.*, LTE, Wi-Fi.

For the evaluation, we consider that the source offers 5 videos for streaming, each one composed of 50 video segments with a duration of 2 seconds each, *i.e.*, in total, each video has a duration of 100 seconds. The video segments are available in three different representations, $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$ with bitrates $\{1750kbps,\ 3000kbps,\ 5800kbps\}$, respectively. These values for the representations and bitrates are according to the values that had been used by Netflix [15]. As presented in Section III-A, the content objects (*i.e.*, the video segments in our evaluation scenario) are divided into Data packets and generations, in order to implement network coding. In particular, for the representations $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$, each video segment is divided into $\{359, 615, 1188\}$ Data packets of 1250 bytes each, and $\{4, 7, 12\}$ generations, respectively. Thus, in total, the source stores $540,500$ Data packets. All the routers are equipped with content stores able to cache between 0.9% and 2.3% of the total Data packets available at the source.

The clients randomly choose a video to request and start the adaptive video retrieval process at a random time during the first 5 seconds of the simulation. The network coding operations are performed in a finite field of size 2^8. The clients use the *dash.js* adaptation logic [24] to choose the representation that better adapts to the current conditions, *i.e.*, the measured goodput and the number of buffered video segments.

B. Benchmarks

We compare the performance of our caching algorithm with the following benchmarks:

- *LCE-NoLimit* — The placement policy is Leave Copy Everywhere (LCE). We assume that the CSs of the routers have enough space to store all the videos.
- *LCE+LRU* — The placement policy is LCE, while the eviction policy is Least Recently Used (LRU), which evicts Data packets with the least recently requested name.
- *NoCache* — In this setting, the routers do not have a CS, *i.e.*, all the Data packets should be retrieved from the source.

C. Evaluation Results

We first evaluate the average cache-hit rate at the routers. In Fig. 4, we can see that by using the PopNetCod caching policy, the routers achieve a higher cache-hit rate than with LCE-LRU. This is because with PopNetCod the number of Data packets cached for a certain name prefix increases smoothly, according to the popularity. In comparison, with LCE+LRU all Data packets received by the router are cached, and the least recently used are evicted from the CS when the capacity is exceeded. Thus, if a router receives Data packets that are requested by a single client, the router still caches them, wasting storage capacity that could be used to cache more popular Data packets that are requested by multiple clients. We can also see that the LCE+NoLimit caching policy defines an upper bound to the cache-hit rate at the routers, since caching all the Data packets with unlimited CS capacity represents the best caching scenario. On the contrary, the NoCache case, where the routers do not have CS capacity, defines a lower bound to the cache-hit rate. Note that in our evaluation the NoCache policy has a non-zero cache-hit rate because our measurement of cache-hit rate also includes Interest aggregations, which is what is being measured in this case.

The increased cache-hit rate that the PopNetCod caching policy brings to the routers has two major consequences: *(i)* the goodput at the clients increases, which enables the adaptation logic to choose higher quality representations when bandwidth is sufficient, and *(ii)* the source receives less Interests, meaning that its processing and network load is reduced.

Let us first evaluate the impact that the increased cache-hit rate at the routers has for the clients. In Fig. 5, it is shown that by using PopNetCod, the clients benefit from an increased goodput, compared to the LCE+LRU policy. This is a consequence not only of the increased cache-hit rate in the network, but also because PopNetCod caches the most popular content in the network edge, which reduces the content retrieval delay. The percentage of video segments delivered to the clients for each of the available representations (*i.e.*, $480p$, $720p$, and $1080p$) with the PopNetCod and LCE+LRU caching policies is shown in Figs. 6 and 7, respectively. We can see that, compared to the LCE+LRU policy, with the PopNetCod caching policy a higher percentage of video segments are delivered in the highest representation available, *i.e.*, $1080p$. This happens because the Data packet retrieval delay is reduced, since more Interests are being satisfied from the routers' content stores, which increases the goodput measured by the clients. The percentage of video segments delivered to the clients in each of the available representations with the upper bound LCE+NoLimit caching policy can be seen in Fig. 8.

Finally, we analyze the impact that the increased cache-hit rate in the routers has for the sources by measuring the load reduction at the source. This metric measures the percentage of Data packets received at the clients that have not been directly provided by the source. It is computed as $1 - N_{\mathcal{S}}^{sent}/N_{\mathcal{C}}^{rcvd}$, where $N_{\mathcal{S}}^{sent}$ denotes the total number of Data packets sent by the source, and $N_{\mathcal{C}}^{rcvd}$ denotes the total number of Data packets received by all the clients. In Fig. 9, we can see that by using the PopNetCod caching policy, the source load is reduced by up to 10% more than by using LCE+LRU, when the CS size is 12.5K Data packets. Note that the load reduction on the source in the NoCache scenario is larger than 0, even if no Data packet is being served from the CSs. This is because the Interest aggregation at the routers makes it possible to serve multiple Interests with the same Data packet, reducing the number of Data packets delivered by the source.

VIII. Conclusions

In this paper, we have presented PopNetCod, a popularity-based caching policy for data intensive applications communicating over network coding enabled NDN. PopNetCod is a distributed caching policy, where each router aims at increasing its local cache-hit rate, by measuring the popularity of each content object and using it to determine the number of Data packets for each content object that it caches in its content store. PopNetCod takes cache placement decisions when Interests arrive at the routers, which naturally enables edge caching. The

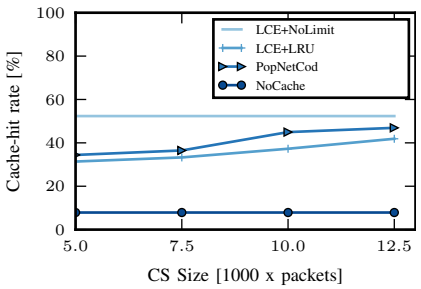


Fig. 4. Average cache-hit rate in the routers.

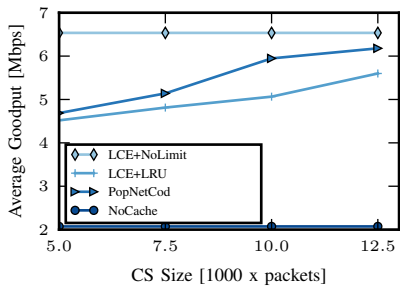


Fig. 5. Average goodput perceived by the clients.

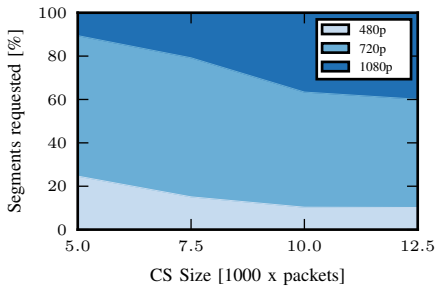


Fig. 6. Percentage of video segments delivered in each of the representations, with PopNetCod.

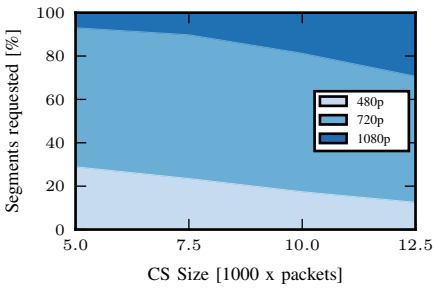


Fig. 7. Percentage of video segments delivered in each of the representations, with LCE+LRU.

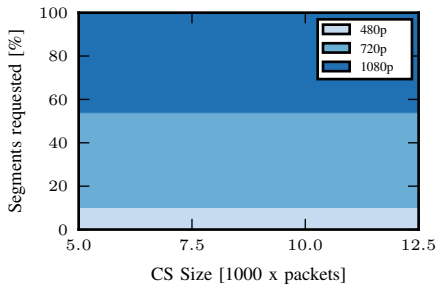


Fig. 8. Percentage of video segments delivered in each of the representations, with LCE+NoLimit.

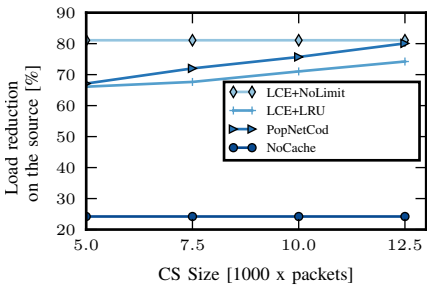


Fig. 9. Load reduction in the source, measured as the percentage of Data packets provided by caches.

evaluation of the PopNetCod caching policy is performed in a Netflix-like video streaming scenario. The results show that, in comparison with a caching policy that uses the LCE placement policy and the LRU eviction policy, PopNetCod achieves a higher cache-hit rate. The increased cache-hit rate reduces the number of Interests that the source should satisfy, and also increases the goodput seen by the clients. Thus, our caching policy presents benefits for the content providers, by reducing the load of its servers and hence its operative costs, and for the end-users, who are able to watch higher quality videos.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," White Paper, Cisco Systems Inc., Jun. 2016.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comp. Comm. Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT'09*, Dec. 2009.

[4] A. Dabirmoghaddam, M. Mirzazad-Barijough, and J. J. Garcia-Luna-Aceves, "Understanding Optimal Caching and Opportunistic Caching at the Edge of Information-Centric Networks," in *Proc. ACM ICN'14*, 2014.

[5] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: incrementally deployable ICN," in *Proc. ACM SIGCOMM'13*, 2013.

[6] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *Proc. ACM CoNEXT'14*, 2014.

[7] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding," in *Proc. ACM NoM Workshop*, Jun. 2012.

[8] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "NetCodCCN: a network coding approach for content-centric networks," in *Proc. IEEE INFOCOM'16*, Apr. 2016.

[9] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[10] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. on Multimedia*, vol. 19, no. 10, Oct. 2017.

[11] A. Ramakrishnan, C. Westphal, and J. Saltarin, "Adaptive video streaming over ccn with network coding for seamless mobility," in *Proc. IEEE ISM'16*, Dec. 2016.

[12] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-coded caching-aided multicast for efficient content delivery," in *Proc. ICC'13*, 2013.

[13] J. Wang, J. Ren, K. Lu, J. Wang, S. Liu, and C. Westphal, "An optimal cache management framework for information-centric networks with network coding," in *Proc. IFIP Networking'14*, Jun. 2014, pp. 1–9.

[14] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: an updated NDN simulator for NS-3," NDN, Tech. Rep. 28, Nov. 2016.

[15] A. Aaron, Z. Li, M. Manohara, J. D. Cock, and D. Ronca, "The Netflix tech blog: Per-title encode optimization," https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2, Dec. 2015.

[16] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: a glimpse at the internet ecosystem through the lens of the netflix cdn," *arXiv preprint arXiv:1606.05519*, Jun. 2016.

[17] "The Netflix ISP Speed Index," Netflix Inc., Dec. 2016. [Online]. Available: https://ispspeedindex.netflix.com/

[18] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A Framework for Joint Dynamic Forwarding and Caching in Named Data Networks," in *Proc. ACM ICN'14*, 2014.

[19] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE INFOCOM'16*, Apr. 2016.

[20] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE INFOCOM'13 Workshops*, Mar. 2012.

[21] N. Abani, G. Farhadi, A. Ito, and M. Gerla, "Popularity-based partial caching for information centric networks," in *Proc. MedHocNet'16*, 2016.

[22] Q. Wu, Z. Li, and G. Xie, "CodingCache: multipath-aware CCN cache with network coding," in *Proc. ACM ICN'13 Workshop*, Aug. 2013.

[23] P. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Sig. Proc. Mag.*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[24] C. Timmerer, M. Maiero, and B. Rainer, "Which adaptation logic? An objective and subjective performance evaluation of http-based adaptive media streaming systems," *arXiv preprint arXiv:1606.00341*, Jun. 2016.

NEST: Efficient Transport of Data Summaries over Named Data Networks

Karim Khalil, Azeem Aqil, Srikanth V. Krishnamurthy
University of California Riverside
{karimk, aaqil001, krish}@cs.ucr.edu

Tarek Abdelzaher
University of Illinois at Urbana Champaign
zaher@illinois.edu

Lance Kaplan
Army Research Lab
lance.m.kaplan.civ@mail.mil

***Abstract*—In many emerging data retrieval applications, in response to queries, consumers are interested in getting a summarized version of content quickly rather than retrieving all available data. Recently, Named Data Networks (NDN) have been considered for efficient transfer of summarized information, but the research is still in its infancy. In this paper, we propose *NEST*, a novel transport protocol for delivering extractive summaries of a dataset distributed across multiple producers over NDN. The goal is to exploit diversity in network conditions between a consumer and different producers towards delivering the consumer-specified summary while minimizing latency. *NEST* first creates a unified hierarchical representation of the available distributed content using state-of-the-art distributed clustering. Then, using this representation of the dataset, the protocol creates interest messages based on which consumers can opportunistically retrieve representative data objects from the best producers while adapting to dynamic network conditions by capitalizing on the flexibility offered by the NDN infrastructure. We implement *NEST* on the Mini-NDN network emulator and evaluate its performance using datasets collected from Twitter. Our experimental results show that *NEST* takes advantage of producer diversity achieving large latency reduction gains of up to 50% compared to baseline protocols.**

I. Introduction

With the emergence of Internet of Things (IoT) and dissemination of online social content, sources of various kinds continuously generate streams of data. The number of such sources is growing continuously, leading to an exponential growth in the available data for a consumer [1], [2]. In fact, consumers may experience a data deluge leading to information overload if they get all the data pertaining to a subject of interest [3]. One way to cope with this data deluge, is via data summarization services which enable clients (whether humans or computer systems) to retrieve summaries with user-specified granularity. These summaries can then be used in analysis and decision making processes.

To exemplify the above, consider a smart city scenario [4] wherein sensors continuously gather data about traffic conditions. On their path to the destination, smart cars contact road infrastructure hot-spots for updates. In this scenario, collected data may have significant redundancy, local data at different repositories have semantic overlap, and network conditions are diverse. Consumers are likely to be interested in receiving content with varying granularity of detail as quickly as possible. As a second example, consumers interested in getting a summary of top stories from a variety of news media may be interested in quickly retrieving only an overview, based on which they may then choose to get more details only on certain stories. Data summarization can be an effective solution in delivering the right level of detail to consumers. In general, summaries can either be processed content that provide a synopsis of the data, or a set of representative samples that sufficiently satisfy the consumer's need. In this paper, we focus on the latter.

There is a set of challenges that will need to be addressed in order to deliver summaries from a set of producers to consumers. First, retrieving summaries from different producers independently will create redundant content, thus wasting communication resources and defying the purpose of summarization. Solving this problem requires the efficient creation of a global representation of the content available at the different producers. Furthermore, the network must be able to match a consumer's request with what is available at the various producers and retrieve the proper summary. In many applications, consumers are not interested in the source of the content or where it is, but rather the content itself and how fast it can be retrieved. Finally, since the network conditions between the consumer and the plurality of producers can be diverse (e.g., varying bandwidth and link delay), the transport framework has to be intelligent to retrieve the content from the "best" producer, i.e., the content that fulfills the consumer's requirement with the best performance (e.g., minimum latency).

In this paper, we develop NDN-based Efficient Summary Transport (*NEST*), a transport protocol which efficiently transfers an extractive summary of a dataset distributed across multiple connected producers to the requesting consumers, with low latency. Our framework is developed on top of the Named Data Networks (NDN) infrastructure, an implementation of the Information Centric Networks (ICN) paradigm. NDN is a pull-based network architecture which supports the forwarding of content from producers to consumers using *hierarchical names*. It offers a way to seamlessly map content to interests and thus, we argue that it is natural to leverage its abilities towards achieving the efficient transport of content summaries from a diverse set of producers to consumers. *NEST* allows producers to converge to a *common namespace*, wherein objects that are very similar are *named* similarly, linking the summarization problem to NDN's name-based forwarding. *NEST* is designed as an end-to-end protocol that runs at the hosts and does not require changes to the underlying NDN infrastructure.

ISBN 978-3-903176-08-9 2018 IFIP

NEST first creates a global hierarchical representation of the dataset by synchronizing the producers' local datasets with minimal overhead. Subsequently, this representation is used to generate an ordered list of names that is sent to interested consumers to guide them in retrieving content summaries of varying granularity. In this list, object names are "constructed" such that, from a set of similar objects at different producers, an object is seamlessly returned from the producer with the most favorable network conditions for each request based on the ordered list, thereby achieving minimum latency. In building *NEST* we make the following key contributions:

- We develop a distributed synchronization algorithm that capitalizes on recent advances in distributed clustering to create a global view of the shared dataset, towards realizing summaries of varying granularity.
- We develop interest-name design rules that automatically and opportunistically adapt to varying network conditions to minimize latency in delivering summaries to consumers over the underlying NDN.
- We implement *NEST* on Mini-NDN, a network emulator for NDN. We then perform extensive evaluations using datasets collected from Twitter. Our results show that *NEST* exploits producer diversity to reduce latency by up to 50% compared to baseline summary transport strategies that retrieve specific data objects.

II. Background

ICN has become popular recently for presenting an alternative and future architecture for the Internet as it becomes more content-centric rather than host-centric, and NDN [5] is one typical implementation. In NDN, consumers send interest messages requesting specific content using hierarchical names, where one data message is returned for each interest message. The interest is generated by a consumer to indicate that she seeks to retrieve a matching object. Partial prefix matching is used when checking whether a named object matches an interest name. In addition, intermediate routers employ multi-path forwarding rules to pass interest messages to the next hop until it reaches producers of the named content. Producers then return data messages where the payload is the data object with a matching name. Data messages are forwarded along the *reverse paths* corresponding to that taken by the interest message. Whenever a router receives a data message, the entry for the corresponding interest is removed from its Pending Interest Table (PIT) after it is forwarded. Any subsequent data messages for the satisfied interest are suppressed (i.e., not forwarded). If caching is enabled, intermediate routers keep a copy of the data messages for a specified period of time, and return it for subsequent matching interests from any consumer. A key feature of the interest message format is the exclusion option; consumers use this optional field to specify object name suffixes that they do not want to retrieve for the given name prefix in the interest message.

In our work, we consider the problem of efficient transport of data summaries. For a given dataset $\mathcal{P}$, a summary is a data subset of $\mathcal{P}$ such that each data point in the summary represents a set of similar (we formally define similarity in Section III-A) data points in $\mathcal{P}$. Recent work [6] proposed

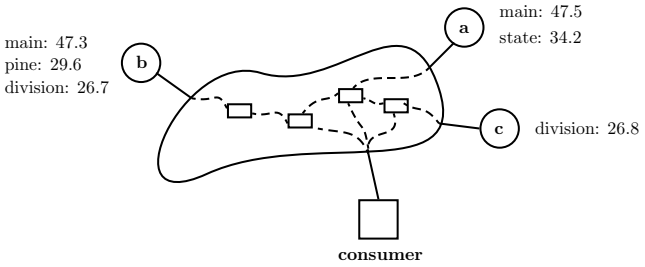


Fig. 1: An example network with three producers a, b and c. Shown are average speed measurements at each producer.

a summary transport protocol for NDN in which an ordered "names list" is created from a hierarchical tree representation of a dataset. The structure of the tree is such that data objects sharing a longer name prefix have more semantic overlap. Thus, when a summary of the content under the tree is requested, returning objects in a shortest-shared-prefix-first order minimizes information loss (relative to retrieving all data) over all different orders of a given summary size by reducing semantic redundancy. Here, as more data objects are transported according to this order, finer granularity details about the data are retrieved. However, only one producer was considered. In data summarization applications in which clients are interested in a summary of the dataset distributed across multiple repositories (producers), dynamic network conditions cause clients to experience very different network delays relative to the different producers from which the summaries are transferred. In addition, redundant content from different producers might be retrieved thereby causing the summaries to be of poor quality (unnecessary redundant content). Thus, when multiple data producers share similar data objects, opportunities to improve latency performance by retrieving *any* object from a set of similar objects are available. However, to exploit these opportunities, a flexible and adaptive data transport protocol is needed. The novelty of *NEST* is that it allows consumers to realize the advantage of *producer diversity* to retrieve summaries with minimum latency while requiring no changes to the underlying NDN architecture.

To illustrate producer diversity, consider an example in which a consumer is interested in a summary of average vehicle speeds in certain section of a city in a given period of time. Measurements are collected from various sensors into a set of three repositories (i.e., producers), named a, b and c, which are connected to the consumer as shown in Fig. 1. In this example, consider data from four streets: main, state, division and pine. Due to varying network conditions (e.g., varying wireless channel quality, network congestion, etc.), the delay in retrieval of data from different producers will be different. In particular, the connection to producer b is experiencing longer delays compared to other producers. Thus, to get a summary of the measurements quickly, the best strategy is to retrieve "main" and "state" from producer a, "division" from producer c and only "pine" from producer b. In other words, the consumer would better retrieve it from the producer with lower delay. The challenge, however, is to figure out which data objects to retrieve from which producer

in order to minimize latency while still fulfilling a notion of completeness of the collected summary. *NEST* offers an efficient solution for this problem be leveraging NDN.

III. System Design

NEST comprises two main functional components viz., (a) Producer Synchronization (ProdSync) and (b) Producer Diversity guided Summary Transport (*PDST*). The first component creates a hierarchical representation of the dataset shared by multiple producers, while the second component manages the transport of data objects between producers and consumers on the NDN. In the following, we discuss the design details of each of the two components.

A. Producer Synchronization

In many applications, data is collected from sensors and cached at a connected set of repositories (i.e., producers) for further processing and dissemination to consumers. Since, transporting objects from individual producers to consumers independently can result in performance penalties and wasteful transfers (e.g., duplicate or redundant data), summarization inherently requires co-ordination or more precisely synchronization between producers. Thus, the first challenge in efficiently delivering summaries from the set of producers (which is typically the order of tens) to consumers is to derive a global view of the available data. This global view necessarily describes the different groups of similar data points (i.e., clusters) as well as the relationship between them. For example, in Fig. 1, measurements from the same street are considered similar and thus are clustered together. Also data from all streets in each neighborhood can be grouped together when only information at a more abstract level is needed by the consumer. When such *hierarchical clustering representation* is available, it suffices to retrieve a representative of each cluster at the required level of detail to get a summary of the available data. This hierarchical representation naturally leads to hierarchical names for each data object based on which cluster it belongs to (similar to naming in Unix-like file systems).

For large datasets, it is not practical to transmit local datasets or large samples thereof to a centralized location to construct such a global hierarchical representation of the available data. In this context, the producer synchronization problem is how to efficiently create a unified view of the hierarchical names of data objects at all the producers. To this end, we develop ProdSync, an iterative distributed clustering algorithm in which producers exchange meta-data of local clusters and samples from their local datasets while incurring minimal communication overhead. This enables the construction of a global tree representation in which each producer maintains information about the position of their local data points in the tree.

To optimize the amount of data exchanged between producers (i.e., overhead) in each iteration of ProdSync, we employ a recently developed distributed clustering algorithm [7], which is based on the construction of ϵ-coresets [8]. This algorithm guarantees a bounded clustering cost relative to a centralized solution, at the minimum communication cost. It was later shown to be communication-optimal [9], where the communication-optimality metric used is the number of data points exchanged between the producers in the network. This metric is also correlated to the convergence time of the ProdSync algorithm; the more the messages exchanged between producers, the more time it takes for ProdSync to converge.

Algorithm 1 ProdSync

Input: Similarity threshold τ, set of producers $\mathcal{N}$
1: Initialize: $\mathcal{L} = r, \mathcal{N}^r = \mathcal{N}$
2: **repeat**
3: Pick an unprocessed tree node $l \in \mathcal{L}$
4: Select coordinator $n_l \in \mathcal{N}^l$
5: Each producer $i \in \mathcal{N}^l$ solves local clustering problem on $\mathcal{P}_i^l$
6: Producers exchange local clustering costs c_i^l
7: Coordinator collects samples $\mathcal{S}_i^l$ from all producers $i \in \mathcal{N}^l$
8: **if** $\max_{p,q \in \cup_i \mathcal{S}_i^l} d(p,q) < \tau$ **then**
9: Coordinator solves global clustering on $\cup_i \mathcal{S}_i^l$
10: **else**
11: l is a leaf node
12: **end if**
13: Coordinator delivers solutions $\mathcal{G}^l$ to all producers in $\mathcal{N}^l$
14: Producers send coordinator local tree info $\mathbf{u}_i^l$
15: Update $\mathcal{T}, \mathcal{N}^l, \mathcal{L}$: $\mathcal{L} \leftarrow g \ \forall g \in \mathcal{G}^l$
16: **until** all $l \in \mathcal{L}$ are processed
Output: Hierarchical tree representation $\mathcal{T}$

Notation: In the following, we introduce notation that will help in the description of ProdSync. Suppose we have a set of connected producers (repositories) $\mathcal{N}$, of size N. Let the global dataset be denoted by $\mathcal{P}$, where $\mathcal{P}_i \subset \mathcal{P}$ is the local subset corresponding to producer $i \in \mathcal{N}$. Let the distance measure between any pair of data points $p, q \in \mathcal{P}$ be given by $d(p,q)$.[1]

Let tree $\mathcal{T}$ be a hierarchical representation for the dataset $\mathcal{P}$, capturing similarities between data points in a hierarchical form. Suppose $\mathcal{L}$ is the set of tree nodes on $\mathcal{T}$, and let $r \in \mathcal{L}$ be the root node. Let $\mathcal{P}^l$ be a data subset of $\mathcal{P}$ holding data under subtree rooted at node l. We also define $\mathcal{P}_i^l = \mathcal{P}_i \cap \mathcal{P}^l$ and $\mathcal{N}^l = \{i \in \mathcal{N} : \mathcal{P}_i^l \neq \phi\}$. Note that $\mathcal{N}^r = \mathcal{N}$ and $\mathcal{P}^r = \mathcal{P}$. In each iteration l of ProdSync, each producer i solves an instance of k-means clustering and sends local information $\mathcal{S}_i^l$ (to be made precise) to a designated coordinator n_l, where the coordinator for tree node r (i.e., n_r), is called the root coordinator. In a clustering problem, the clustering cost is the sum of squared distances between each point in the dataset and its corresponding cluster center. We say that p and q are similar if $d(p,q) < \tau$, for a given threshold τ.

ProdSync details: The details of ProdSync are presented in Alg. 1. The algorithm iteratively clusters the dataset $\mathcal{P}$ shared across all producers $\mathcal{N}$ to generate the tree $\mathcal{T}$. To process a node $l \in \mathcal{L}$, a coordinator n_l is first selected[2] which later collects information about local clustering solutions from all producers in $\mathcal{N}^l$. In each iteration, the goal is to find a set of

[1] Vector space representation of data as well as similarity measures vary depending on application and data type. While we use specific representation and similarity measures in Section V, the effect of these on the clustering quality is of separate interest and is beyond the scope of this paper.

[2] We defer a discussion of how we implement coordinator selection to Section IV.

k cluster centers $\mathcal{G}_l$ at the coordinator, representing all data subsets $\mathcal{P}_i^l, i \in \mathcal{N}_l$. This is achieved by using an efficient distributed clustering algorithm [7], described briefly in the following.

Each producer first solves a k-means clustering problem on the local dataset $\mathcal{P}_i^l$ and then non-uniformly samples $\mathcal{P}_i^l$ based on the clustering costs c_i^l collected from all other producers $i \in \mathcal{N}^l$. In this sampling, a point with higher cost is sampled with higher probability. Each producer constructs its local portion of the ϵ-coreset, $\mathcal{S}_i^l$, which consists of the samples and their corresponding weights. Then, $\mathcal{S}_i^l$ are collected at the coordinator. A weighted k-means clustering problem is solved on the weighted samples $\cup_i \mathcal{S}_i^l$. The solution of the global clustering problem is then shared with producers in $\mathcal{N}^l$.

In ProdSync, a global clustering is solution is computed for a node l only when l is not a leaf node. We reach a leaf node in $\mathcal{T}$ (and hence stop further iterations of clustering on that node) when the diameter of the cluster is less than the threshold τ (condition on Line 8). In this case, each producer then updates the coordinator with cluster membership counts $\mathbf{u}_i^l$, which is a vector of size k. This information helps in creating the tree representation of the dataset as well as names for objects. Then, new nodes are added to the tree $\mathcal{T}$, one node representing each cluster in $\mathcal{G}_l$. Iteration stops when all nodes in $\mathcal{L}$ are processed.

Complexity analysis: At each iteration $l \in \mathcal{L}$, three rounds of message exchanges between the coordinator and other producers are required. First, the costs of local clustering solutions are collected by the coordinator and the sum cost is shared with all producers. Then, samples are sent to the coordinator and the solution $\mathcal{G}_l$ is returned to each producer. Finally, updates $\mathbf{u}_i^l$ are sent to the coordinator. The communication overhead is thus $O(N)$ per iteration.

To process a node $l \in \mathcal{L}$, each producer i solves an instance of the k-means clustering problem on the local dataset $\mathcal{P}_i$ (Steps 5 and 9). This problem is NP-hard [10]. However, there exist efficient approximations such as the Lloyd's algorithm [11], with time complexity $O(|\mathcal{P}_i|ksw)$, where s is the dimensionality of vectors representing the data points and w is the number of iterations needed for convergence. It was shown that, in practice, k-means converges in linear time with respect to the number of data points [12]. The number of nodes on the tree (i.e., $|\mathcal{L}|$), can vary between $O(\log_k |\mathcal{P}|)$ to $O(|\mathcal{P}|)$. In practice, we pipeline and parallelize the processing of tree nodes such that communication delay does not contribute a purely additive component to the total processing time. We study this in detail in Section V.

B. Producer Diversity guided Summary Transport

Given the hierarchical cluster representation of the data (as computed in Section III-A), we now need to decide the order in which data objects must be requested from these clusters. The intuition is that, in constructing a summary, we first want to have at least one representative of each cluster (e.g., a measurement of speed on each street), then get a second representative of each cluster (a second measurement from that street), and so on. If clusters are hierarchical, then we need one representative of each big cluster (say, street) before getting a representative of each sub-cluster (say city block on a street). As illustrated in the example in Section II, the choice of representative to retrieve entails a choice of producer, some being more accessible (better network conditions) than others. We want to retrieve representatives from more accessible producers. A challenge is thus to decide on a retrieval plan that minimizes latency.

In this section, we develop an efficient transport protocol, called Producer Diversity guided Summary Transport (*PDST*) that takes the output tree $\mathcal{T}$ from ProdSync, transforms it to a List of Ordered Names (LON) that is delivered to interested consumers. To construct the LON, $\mathcal{T}$ is parsed such that leaves are visited in certain order and a data point is chosen from visited leaf and then added to LON. The tree is traversed such that the marginal utility of retrieved data objects is maximized. Thus, as more items are retrieved as per the LON, a more fine-grained summary is obtained by the consumer. Consumers request summary data objects based on the LON, and *PDST* delivers data objects from producers to consumers with minimum transport latency, defined as follows.

Definition 1. *The transport latency $T(j)$ corresponding to a given interest message j is the total time delay between sending the interest message and the reception of a data message.*

Fix an interest message j and let $\mathcal{N}(j) \subset \mathcal{N}$ be the set of producers with data objects that can satisfy interest message j. Let $T_i(j)$ be the transport latency when data is returned from producer i. The objective of *PDST* is to minimize the latency in retrieving data objects, whenever matching data objects are available at multiple producers. In other words, PDST aims to achieve $T^*(j) = \min_{i \in \mathcal{N}(j)} T_i(j)$. While doing so, the protocol must adapt to dynamic network conditions, and specifically varying link delays.

Satisfying the minimum transport latency and adaptability to dynamic network conditions, are challenging problems for multiple reasons. First, it is undesirable that consumers maintain state information for all available producers (e.g., by keeping the history of received data). Moreover, explicitly and continuously measuring transport latency for objects from different producers will incur non-negligible overhead. The novelty of *PDST* is that it achieves the aforementioned goals by crafting interest messages in a format that capitalizes on the features of NDN. Specifically, *PDST* exploits NDN's forwarding characteristics viz., multi-path and partial prefix match forwarding. It also leverages the fact that intermediate routers suppress multiple data messages retrieved in response to a single interest message and only forward the first match. The main observation is that if $\mathcal{N}(j)$ always reflects producers that have similar data objects that satisfy j, NDN operations will automatically help achieve $T^*(j)$ without the need to explicitly measure network conditions. To this end, *PDST* does not require any modifications to NDN and is only run at the producers and consumers as an application.

PDST achieves minimum transport latency and adapt to varying network conditions using three different processing steps at the different participating network entities. Below, we discuss the different steps of *PDST* in detail before we

formalize our result.

1) Root-coordinator-side PDST: The root coordinator is tasked with generating the LON from $\mathcal{T}$. First, names for all objects are created. These names are then processed to identify producer diversity opportunities. Finally, the tree is traversed to generate the ordered list of names.

First, names of all data objects at the leaves of $\mathcal{T}$ are automatically created. In particular, during clustering, tree nodes are given labels (e.g., '0' for the left branch and '1' for the right branch when $k = 2$), and data objects at the leaves are named by concatenating label names from the root node to the leaf, similar to the work in [13], thereby constructing a name prefix. However, unlike the work in [13], the root coordinator n_r does not have actual data points from all other producers; rather, it has the *counts* of data objects under each leaf from each producer. This information is collected in Step 14 of Alg. 1. Thus, n_r can now generate names for all data objects at the leafs of $\mathcal{T}$.

Consider a tree $\mathcal{T}$ created using ProdSync with $k = 2$ for data at two producers a and b. Under some tree leaf l' with a name prefix $p = /t/0/0/1/0/1$, suppose producer a and producer b have two and three data objects, respectively. Here, t represents the *topic* at the root of the tree. Now, the root coordinator can simply name data objects under this leaf as $/t/0/0/1/0/1/a0$, $/t/0/0/1/0/1/a1$, $/t/0/0/1/0/1/b0$, $/t/0/0/1/0/1/b1$, $/t/0/0/1/0/1/b2$. Based on the user-specified and application-dependent similarity measure, objects under l' are deemed similar. At the same time, each of the producers a and b will fix some order for their local data objects, based on user-specified weights corresponding to each data object (e.g., content popularity, freshness, etc). Note that while each producer can rank order similar local data objects using user-specified weights, the relative ordering between similar data objects at different producers is not needed at the root coordinator. This is because our design gives priority to improving transport latency by retrieving the next (highest weight) data object (based on local ranking) from the producer with the best network conditions.

Denote any leaf of $\mathcal{T}$ with objects from multiple producers, as an *opportunity leaf*. The next step for the root coordinator is to process the data object names such that a special symbol (˜) is concatenated at the end of all object names under any tree leaf where data objects belonging to multiple producers exist. Thus, for leaf l', the object names will be processed to be $/t/0/0/1/0/1/a0$˜, $/t/0/0/1/0/1/a1$˜, $/t/0/0/1/0/1/b0$˜, $/t/0/0/1/0/1/b1$˜, $/t/0/0/1/0/1/b2$˜. This special symbol in the object names will later be used by the consumer to construct interest messages that allow retrieval of data objects from the producer with the minimum transport latency.

Given the hierarchical tree representation $\mathcal{T}$ created using ProdSync as described in Section III-A, the root coordinator can now transform $\mathcal{T}$ into an LON by traversing $\mathcal{T}$ from the root to the leaves and returning the name of the object at the leaf. During traversal the branches are selected such that an object with the shortest-shared-prefix, with respect to previously returned object names, is returned.

Finally, this processed list is sent to the consumers upon request. In particular, when a consumer requests a summary of a dataset under the prefix $/t$, the root coordinator will send a data message carrying the LON for data objects under the corresponding tree. Note that the LON is generally of much smaller size compared to data objects (e.g., in social media, a tweet could have an image or video object embedded in it).

2) Consumer-side PDST: Each consumer running the *NEST* application will first request the LON under some tree root $/t$. The consumer then sends interests for items in the LON in the given order. However, the interest names used will vary depending on whether the object belongs to an opportunity leaf. For such objects, a *producer diversity opportunity* exists and thus the consumer can take advantage of it. In particular, for any object name in the LON ending in ˜, the consumer sends the interest message with the partial name up to the prefix of the corresponding leaf in $\mathcal{T}$. For example, if the next data object name in the LON is $/t/0/0/1/0/1/b0$˜, the consumer sends the interest message $/t/0/0/1/0/1/$ instead.

This interest will be forwarded by the underlying NDN to all producers with data objects under the corresponding opportunity leaf. Thus, all producers will respond with data objects under the given leaf, and only one data message will be forwarded to the requesting consumer while other messages will not be forwarded. To avoid retrieving duplicate objects that were retrieved previously using the same partial name, the consumer employs the *exclude* option in the interest message. In particular, it includes the last component in the name of the objects retrieved previously under same leaf. For example, when an interest message is sent with the partial name $/t/0/0/1/0/1/$ and data object $/t/0/0/1/0/1/b0$ is retrieved, the next interest message for an object belonging to the same opportunity leaf will be $/t/0/0/1/0/1/(-b0)$.

One of the main advantages of crafting the interest messages as described above is that the framework automatically adapts to changing link delays. Thus, two consumers sending the same interest message will get potentially different (but semantically similar) data objects from different producers. Furthermore, as network conditions change over time, a consumer may get data objects from other producers because of latency advantages. Since *PDST* capitalizes on NDN forwarding rules, it also works when caching at intermediate routers is enabled without the need to modify the software they run. Specifically, caches will also use partial prefix matching and return objects with matching names. If no matches are found in the cache, the interest will be further forwarded.

3) Producer-side PDST: On the producer side, each producer maintains an ordering of the local data points based on user-specified and application-dependent weights. For example, in a social media application, the popularity of the content could be used as the weight. In a sensor network application, more recent events or measurements could be weighted highly if freshness is desired. Whenever the producer receives an interest message with a partial name, it responds with a data object from the subtree specified by the name prefix, returning the first object in order after the excluded objects. For example, if the interest message received by producer b is $/t/0/0/1/0/1/(-b0)$, then it returns object $/t/0/0/1/0/1/b1$. Note that the object name $b1$ might not be the actual object name at producer b, but rather a pointer

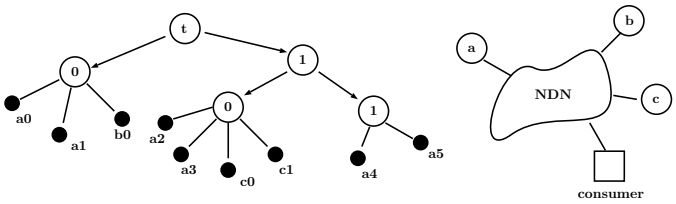


Fig. 2: Example network and tree with three producers (a,b,c) and a consumer.

	NEST's LON	Interest name sent	Data name received
1	$/t/0/a0^{\sim}$	$/t/0/$	$/t/0/b0$
2	$/t/1/0/a0^{\sim}$	$/t/1/0/$	$/t/1/0/c0$
3	$/t/0/a1^{\sim}$	$/t/0/(-b0)$	$/t/0/a0$
4	$/t/1/1/a4$	$/t/1/1/a4$	$/t/1/1/a4$
5	$/t/0/b0^{\sim}$	$/t/0/(-b0,-a0)$	$/t/0/a1$
6	$/t/1/0/a3^{\sim}$	$/t/1/0/(-c0)$	$/t/1/0/c1$
7	$/t/1/1/a5$	$/t/1/1/a5/$	$/t/1/1/a5$
8	$/t/1/0/c0^{\sim}$	$/t/1/0/(-c1)$	$/t/1/0/a2$
9	$/t/1/0/c1^{\sim}$	$/t/1/0/(-c1,-a1)$	$/t/1/0/a3$

TABLE I: Example of *PDST* operation.

to a data object under the given prefix which is second in order based on the weights. This order is maintained only locally by each producer.

In Table I, an example LON is shown. The corresponding network with one consumer and three producers, as well as the hierarchical data representation $\mathcal{T}$ are shown in Fig. 2. In the example network, the transport latency to producer c is the lowest, then to producer b, and then to producer a. The second column is generated by the root coordinator in *NEST* and represents the LON delivered to consumers requesting a summary of content under the prefix $/t$. In the third column, the interest names that the consumer uses in interest messages are shown. The names of the data objects received by the consumer in response, are shown in the last column. Note that the interest names in the third column are adapted based on names of data objects received so far (i.e., from previous rows), as listed in the last column. For example, consider row number 6. Here, the interest sent is for a data object that is under the prefix $/t/1/0/$. Since the consumer previously received the object $/t/1/0/c0$ (in row 2), it now includes $c0$ in the exclude field of the interest message. The NDN forwarding will pass the interest message $/t/1/0/(-c0)$ to all producers, but it will reach producer c first since it has the best network conditions with respect to the consumer. Now, producer c will check its local dataset for data objects under prefix $/t/1/0/$ and with rank order subsequent to object $c0$, returning object $/t/1/0/c1$. Data objects returned from other producers will then be suppressed by intermediate routers since the interest message would have been already satisfied by object $/t/1/0/c1$.

In the following, we formalize our main result. The proof is omitted for brevity.

Proposition 1. *Fix an interest message j. PDST achieves minimum latency $T^*(j)$.*

We note that Proposition 1 implies that *PDST* minimizes latency even if the link delay varies while the interest or data message has not been received at the destination.

Pipelining interests: PDST uses an adaptive pipelining window, which controls how many pending interests are allowed at any given time. In addition to being limited to a maximum size W, the window size is adapted based on the LON and the progress made thus far in processing the list. In particular, the consumer can send interests from the LON until a new entry requires sending a partial name which is *already* in use in a pending interest, or until the maximum window size is reached, whichever is smaller. This design prevents retrieval of duplicate objects, since names of previously retrieved objects are added to the exclude fields of subsequent interests, with the same partial names. We evaluate the choice of W in Section V. We also note that loss management is handled by the underlying NDN mechanisms through the use of timeout timers and retransmissions.

Caching: Before we conclude this section, we discuss how caching affects the performance of our system. As the number of consumers increase, it is expected that caches at intermediate routers will return data objects more often, improving latency performance with respect to a scenario wherein caching is disabled. This in turn could reduce the producer diversity opportunities that *NEST* tries to exploit to improve performance; the data is already cached en route. However, as will be shown in Section V, the marginal gain in latency reduction is large even when caching is enabled. In addition, the combined gain is substantial.

IV. Implementation

We implement *NEST* on Mini-NDN [14], an NDN network emulator based on the popular Mininet [15] virtual network environment. In Mini-NDN, a network topology is specified in which nodes are connected via links parameterized by link delay, bandwidth as well as loss percentage. Each node in the network is capable of running NDN applications, forwarding NDN packets according to the specified routing policy, as well as caching forwarded content.

Mini-NDN accomplishes these NDN functionalities by running an instance of Named Data Link State Routing Protocol (NLSR) [16] and NDN Forwarding Daemon (NFD) [17] on each instantiated node in the network. NLSR is a routing protocol responsible for populating NDN's Forwarding Information Base (FIB) while NFD is a network forwarder that is fully capable of forwarding NDN packets according to a diverse set of routing strategies.

Since Mini-NDN emulates the actual operations of NDN networks, one primary advantage is that the applications developed and tested on Mini-NDN can be readily operational on the NDN testbed [18] or other actual NDN networks.

A depiction of *NEST*'s different components is shown in Fig. 3. Each producer in the network runs the two functional components of *NEST* (ProdSync and *PDST*) simultaneously while the consumer runs *PDST*. First, producers in *NEST* run the "*NEST* Sync" application, which is responsible for implementing ProdSync and creating "*NEST* tree". In our implementation, we use k-means clustering with $k = 2$. The *NEST* tree is then passed to the "*NEST* Prod" application. In

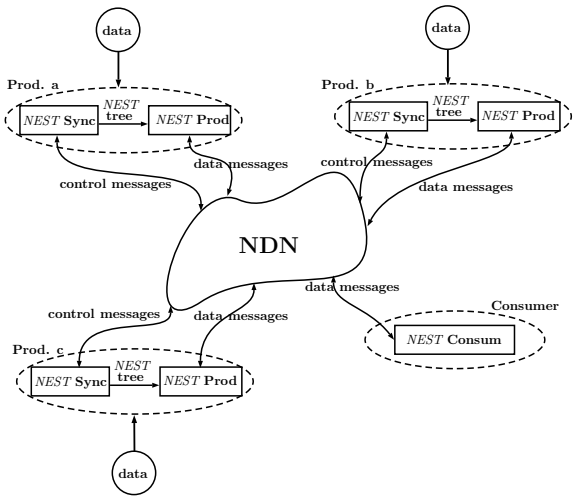


Fig. 3: A Network with three producers and a consumer running *NEST*. Each box represents an application running on producers or consumers.

this application, the tree is transformed to an LON which is then used to guide the transport of data items. Finally, the third application is the "*NEST* Consum" application running at each consumer. This component is responsible for sending interest messages that are responded to by the *NEST* Prod application running at each producer.

We implemented all the applications in Python. In the *NEST* Sync application, clustering information of every tree node is kept in a data structure that holds the state of the computations and data exchanged between the coordinator and non-coordinator producers. State and data are encoded into control messages, where an interest control message requests the start of computation or delivers the notification that a computation is completed, while the corresponding data control message delivers an acknowledgment or the requested data. On the other hand, in the *NEST* Consum application, the consumer implements a pipelining window of pending interests and a method to transform the LON to interest messages with partial names. *NEST* Prod implements a partial interest name match function to select messages to be sent to the consumers.

V. Evaluation Results

Setup: Our evaluations are based on a dataset collected from Twitter over a period of time from Dec 2016 to Mar 2017 using Twitter's streaming API and a set of search keywords for trending topics in politics, sports, and entertainment. Overall, we use a dataset of about 80K tweets in our evaluations.

In each experiment, we randomly distribute a sample of the dataset uniformly across the set of producers. We first pre-process the collected tweets to remove stop words, special characters, links and attachments, producing tokens. These tokens are then transformed to a high dimensional vector representation by computing the product of term frequency and inverse document frequency (*tf-idf*) [19], a popular method for text vectorization. We use the `sklearn` library [20] vectorizer to achieve this task.

TABLE II: ProdSync convergence time.

N	**3**	**5**	**7**	**9**
Time(s)	34	38	83	122

In Mini-Net, links connecting the producers and consumers are characterized by the link delay, the bandwidth, and the message loss rate. In our experiments, we fix the bandwidth and loss rates, and vary the link delays. We note that in Mini-Net, each host in the network runs the NDN stack and thus can be used as a producer, a consumer, and a forwarding switch, simultaneously. In addition, hosts have content stores and thus can cache data objects. In our experiments, we use a network topology similar to that used in the NDN testbed [18] and we have a varying number of producers consumers for different experiments as will be discussed in the following.

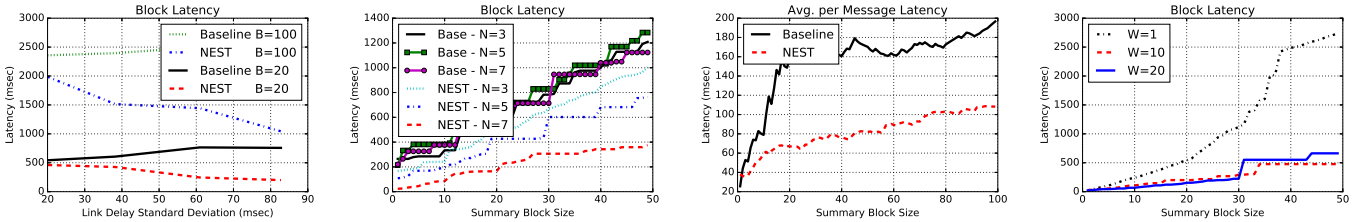
In the following, we define terms that we use in our evaluations. Let the *summary block* with size B be the number of data objects the consumer has to fetch in order to have a satisfactory summary. The *block latency* t_B is the delay from sending the interest message for the first data object in the block, until the successful reception of the data message corresponding to the last data object in the summary block. This quantity is directly proportional to the per interest latency defined in Section III-B.

We divide the evaluation results into two parts. In the first, we evaluate the performance of the ProdSync algorithm and quantify performance in terms of the convergence time. In the second, we focus on the latency performance of *PDST* and compare it to a baseline summary transport protocol with no producer diversity (i.e., a system in which the LON is used to retrieve data objects from specific producers, similar to the protocol in [6]).

A. Producer Sync

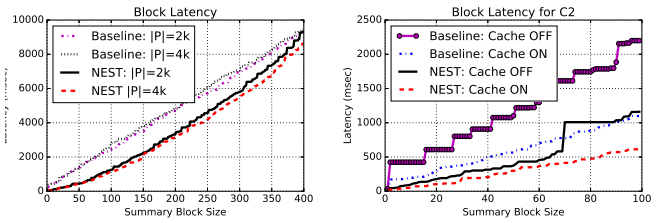
We consider networks with different numbers of producers and distribute a dataset of $4000N$ tweets uniformly at random over the N producers. We use Euclidean distance to measure similarity between different vectors representing tweets, and use a similarity threshold $\tau = 0.9$ as the stopping criterion for ProdSync. For the coordinator selection, in each iteration, we let the producer with the smallest ID perform the coordination tasks for the corresponding tree node. Producers are connected with link delays of 10 milliseconds.

We first evaluate ProdSync's convergence time. For scenarios with $N = 3, 5, 7, 9$ producers, we repeat the experiment 10 times and report the average convergence time in each case. Table II outlines the results. It can be seen that ProdSync's convergence time is approximately linear in the number of producers and the dataset size for $N > 3$. In many applications (such as traffic monitoring, news stories updates), major dataset changes happen on the order of hours. Thus, ProdSync provides a practical means for constructing the global representation of the distributed dataset as it can be run periodically at a rate that is faster than the rate of data evolution.



(a) Link delay variance. (b) Number of producers. (c) Per message latency. (d) Pipelining window sizes.

Fig. 4: Latency performance.



(a) Different sample size $|\mathcal{P}|$. (b) Caching.

Fig. 5: Latency performance.

B. Latency Performance

In this section, we evaluate the latency performance of *PDST* after the LON has been delivered to the consumers. We compare the performance to the baseline protocol.

1) Link delay variance: First, we vary link delay variance and measure the incurred latency. We fix the maximum pipelining window size to $W = 10$ and vary the link delays from the producers to the consumer in the range 5 to 200msec while maintaining a fixed average. Here, we consider a topology with 5 producers and 1 consumer, and we consider two different summary block sizes $B = \{20, 100\}$. As shown in Fig. 4a, the block latency improves as the link delay variance increases. This is because *NEST* effectively checks if similar objects exist at producers with better network conditions and fetches objects from those producers first. Essentially, objects with slow retrieval times are pushed to the end of retrieval order. Compared to the baseline system, block latency performance is improved by more than 40% when the link delay standard deviation is 50msec.

In Fig. 4b, we plot the block latency t_B for a varying B. We also show the performance for topologies with different number of producers. First, we observe that while the baseline system performance does not change when number of producers change, *NEST* fully utilizes producer diversity. In particular, as the number of producers increases, diversity improves and block latency decreases.

We also study the per message latency when $N = 5$, where W and link delays are chosen as in previous experiments. In Fig. 4c, we plot the average per message latency vs. different B. The figure shows that by taking advantage of producer diversity, the latency improvements can be as high as 50%. Note that as the block size increases for a fixed dataset size, this gain is expected to decrease. We study the effect of the ratio $\frac{B}{|\mathcal{P}|}$ in Section V-B3.

2) Pipelining: Next, we study the effect of the maximum pipelining window size W on the block latency. In Fig. 4d, there are five producers with link delays similar to those in the previous experiments. Note that *PDST*'s adaptive pipelining does not send new interest messages while pending interests with the same partial name exist, to avoid duplicate object retrievals. It is seen that pipelining improves block latency compared to a simple stop and wait approach ($W = 1$). In addition, consumers will experience more packet losses as W increases and thus more bandwidth wastage. The figure shows that diminishing gains result due to increasing W. We find that $W = 10$ achieves the best latency performance.

3) Impact of dataset size: Next, we study the effect of the dataset size on the block latency. It is expected that as the ratio $\frac{B}{|\mathcal{P}|}$ decreases, the latency gain of *NEST* increases. In other words, when the requested summary block size B is comparable to the dataset size, consumers may not be able to avoid fetching objects from producers with unfavorable network conditions. We fix five producers with link delays similar to previous experiments. Fig. 5a shows that, for a given block size B, the latency reduction gain is only slightly reduced when the sample size is halved. When $|\mathcal{P}| = 2000$, *NEST* can achieve a positive gain for summary block sizes up to 20% of the dataset, which is reasonable for applications in which consumers are interested only in data summaries of large datasets.

4) Caching: Finally, we study the performance of *NEST* when caching is enabled in the underlying NDN. In this experiment, the topology has two consumers and five producers with similar link delays as in previous experiments. Both consumers are using the same LON. We introduce a delay between the time each consumer starts fetching items to see the effect of caching. Caching allows intermediate nodes to temporarily store content that was previously forwarded to other hosts in the network. As seen in Fig. 5b, *NEST* yields latency performance improvements of about 50%. Compared to the baseline performance with caching disabled, the combined latency reduction gain is more than 70%.

VI. Related Work

There has been prior work on selectively sending a representative subset of data instead of the entire dataset [21], [22]. However, unlike our work, these efforts are application specific. Moreover, these approaches try to optimize for energy

efficiency in contrast to our goal of minimizing latency in retrieving the summary. Achieving our goal requires an approach that is much different from those proposed in these efforts.

More recently, multiple works considered optimizing latency in NDN [23]–[25]. In these works, architectural changes to NDN are proposed to improve the support for low latency applications such video conferencing [23]. In addition, multipath routing as well as network coding are employed [24], [25] to improve performance of video streaming. Unlike these approaches, *NEST* does not require changes to underlying NDN infrastructure and operates as an end-host application. Thus, we argue that it is much more general and easy to deploy.

On the other hand, distributed dataset synchronization was recently addressed [26]. The goal is to efficiently synchronize the state of a group of hosts for applications such as group text messaging. This is different from the problem we consider wherein we address producer synchronization, since we do not require all hosts to have the full dataset.

The closest works to ours are Espresso [13] and InfoMax [6]. The former creates a tree representation and object names from a given dataset for creating summaries, while the latter transforms the tree into an ordered list for transport. However, their model considers only a single producer. While we use a similar approach towards summarization, we address a different set of challenges wherein the dataset is distributed across multiple producers. In particular, transport performance is our primary issue of focus; this was not addressed in these works.

VII. Conclusion

In this paper, we target the problem of delivering a summary of a large dataset to consumers from a set of producers with low latency. Retrieval of such a summary of the dataset, is becoming popular in many emerging applications. We propose *NEST*, an efficient data summary transport protocol which leverages the NDN architecture towards achieving this goal. Our novel framework opportunistically fetches data from the producers with good network conditions relative to consumers after constructing a global view of the dataset shared between producers and establishing similarity relations between data points. Our experimental results show that large latency reduction gains can be achieved compared to baseline strategies that do not exploit producer diversity. The gains are especially noteworthy (up to 50%) when the number of producers and link delay variations are large.

Acknowledgment: This work was partially supported by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also partially supported by the NSF CPS grant 1544969.

References

[1] L. Columbus, "Roundup of internet of things forecasts and market estimates, 2016," https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#1b7ea093292d, 2016.

[2] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[3] B. Marr, "Big data overload: Why most companies can't deal with the data explosion," https://www.forbes.com/sites/bernardmarr/2016/04/28/big-data-overload-most-companies-cant-deal-with-the-data-explosion/#33cde7b06b0d, 2016.

[4] S. H. Bouk, S. H. Ahmed, D. Kim, and H. Song, "Named-data-networking-based its for smart cities," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 105–111, 2017.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Commun. Review*, vol. 44, no. 3, pp. 66–73, 2014.

[6] J. Lee, A. Kapoor, M. T. Al Amin, Z. Wang, Z. Zhang, R. Goyal, and T. Abdelzaher, "InfoMax: An information maximizing transport layer protocol for named data networks," in *IEEE 2015 24th Int. Conf. on Computer Commun. and Networks (ICCCN)*, 2015, pp. 1–10.

[7] M.-F. F. Balcan, S. Ehrlich, and Y. Liang, "Distributed k-means and k-median clustering on general topologies," in *Advances in Neural Information Processing Systems*, 2013, pp. 1995–2003.

[8] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of Computing*. ACM, 2004, pp. 291–300.

[9] J. Chen, H. Sun, D. Woodruff, and Q. Zhang, "Communication-optimal distributed clustering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3727–3735.

[10] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

[11] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. on Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[12] D. Arthur, B. Manthey, and H. Röglin, "Smoothed analysis of the k-means method," *J. ACM*, vol. 58, no. 5, pp. 19:1–19:31, Oct. 2011.

[13] J. Lee, M. T. Al Amin, and T. Abdelzaher, "Espresso: A data naming service for self-summarizing transport," in *2017 14th Annual IEEE Int. Conf. on Sensing, Commun., and Networking (SECON)*, 2017, pp. 1–9.

[14] "named-data/mini-ndn," https://github.com/named-data/mini-ndn.

[15] "Mininet," http://mininet.org/.

[16] "NLSR - named data link state routing protocol," http://named-data.net/doc/NLSR/current.

[17] "NFD - named data networking forwarding daemon," https://named-data.net/doc/NFD/current.

[18] "NDN testbed," https://named-data.net/ndn-testbed/.

[19] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first Instructional Conf. on Machine Learning*, vol. 242, 2003, pp. 133–142.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[21] H. Gupta, V. Navda, S. Das, and V. Chowdhary, "Efficient gathering of correlated data in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 1, p. 4, 2008.

[22] Y. Ma, Y. Guo, X. Tian, and M. Ghanem, "Distributed clustering-based aggregation algorithm for spatial correlated sensor networks," *IEEE Sensors Journal*, vol. 11, no. 3, pp. 641–648, 2011.

[23] M. Almishari, P. Gasti, N. Nathan, and G. Tsudik, "Optimizing bi-directional low-latency communication in Named Data Networking," *SIGCOMM Computer Commun. Review*, vol. 44, no. 1, pp. 13–19, 2013.

[24] S. de Arco, J. Eduardo, E. Bourtsoulatze, N. Thomos, and T. Braun, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. on Multimedia*, 2017.

[25] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," in *IEEE INFOCOM*, 2017.

[26] Z. Zhu and A. Afanasyev, "Let's Chronosync: Decentralized dataset state synchronization in named data networking," in *2013 21st IEEE Int. Conf. on Network Protocols (ICNP)*, 2013, pp. 1–10.

MUCA: New Routing for Named Data Networking

Chavoosh Ghasemi[1], Hamed Yousefi[2], Kang G. Shin[2], and Beichuan Zhang[1]
[1]Department of Computer Science, The University of Arizona
[2]Department of Electrical Engineering and Computer Science, The University of Michigan

***Abstract*—*Named Data Networking* (NDN) is a fundamental paradigm shift from host-centric to data-centric Internet architecture. Among its numerous benefits, in-network caching and multipath forwarding are two prominent features that can significantly improve the performance and resiliency of networks and applications. The current NDN routing protocols, however, still focus on the traditional problem of forwarding content requests to content producers, without explicit or efficient support of in-network caching and multipath forwarding, which will limit NDN's potential and benefits to applications.**

In this paper, we propose a new intra-domain name-based routing protocol to provide simple and scalable support for MUltipath forwarding and in-network CAching (MUCA). While MUCA collects the network topology and computes the shortest paths to content producers in the same fashion as link-state routing protocols, it also learns multiple alternative paths from neighboring routers similar to distance-vector routing protocols. Moreover, by labeling each route update at the entry point into a network, internal routers select the same border router for the same name prefix, which enhances the hit ratio of cached contents. Our in-depth simulations demonstrate MUCA's effectiveness in reducing content retrieval delay and improving network resiliency while lowering the routing protocol overhead.

I. Introduction

Named Data Networking (NDN) is a clean-slate future Internet architecture and also an important representative of Information Centric Networking (ICN) [22], [23]. In NDN, content is identified by a hierarchical name, and both the requests (i.e., *Interests*) and the responses (i.e., *Data*) carry the content name rather than a source/destination address. Among the various benefits of the NDN architecture, in-network caching and multipath forwarding are two major features that can significantly improve network performance and resiliency. Since each network packet carries a unique name that identifies its content, intermediate routers can cache Data in its returning path to the requester(s) to serve future network Interests, i.e., NDN enables native in-network caching. Moreover, as pointed in [21], NDN's forwarding plane can detect routing loops by itself and choose a different next-hop if loop happens. This allows NDN routers to make use of multiple next-hops and adapt the choice based on content retrieval performance.

Full realization of the potential of in-network caching and multipath forwarding needs support from the underlying routing protocol. (Note that the routing plane in NDN is decoupled from the forwarding plane as discussed in Section II.A.) At the heart of NDN, the forwarding engine needs a routing protocol to efficiently compute and install proper forwarding entries in order to forward Interests towards the corresponding content provider(s). While some studies have been done on NDN routing protocols [10], [17] or similar content-centric routing protocols [8], [9], they all focus on the traditional problem of computing the shortest path towards a content producer, without explicit or efficient support of in-network caching and multipath forwarding.

To increase cache hit ratio in network caches, requests for the same content but generated by different consumers should merge as early as possible in the network, i.e., their forwarding paths merge before they reach the content producer. Traditional shortest-path computation does not take this into consideration, thus the result is opportunistic for caching. To alleviate this deficiency, MUCA labels each routing announcement/update in border routers such that all the internal routers will select the same border router for the same name prefix. This guarantees that requests for the same content will always merge before they go out of the network while improving the chance of their merge even before reaching the border router. Not only does this feature increase the efficiency of in-network caching, but also reduces the transport cost incurred by traffic between networks.

To take advantage of the NDN's capability of multipath forwarding, the routing protocol is expected to provide the forwarding plane with multiple next-hops for each name prefix. Traditional routing protocols only compute a single best path. NLSR [17], a name-based link state routing protocol currently used in the NDN testbed, computes a list of ranked next-hops by running Dijkstra algorithm in a router for each of its active interfaces, which can incur significant computational overhead, especially for routers with high connectivity. To address this issue, MUCA—as a link-state routing protocol—borrows a distance-vector mechanism, retrieving routing tables from neighboring routers instead of computing them. From these retrieved routing tables, one can easily figure out the ranked list of possible next-hops and save CPU cycles for the local router.

In addition to explicit support of caching and efficient support for multipath, MUCA employs a simpler mechanism than NLSR to propagate incremental routing updates. NLSR treats the routing update propagation problem as a data synchronization problem, and adopts ChronoSync [25] to synchronize the link state database (LSDB) between neighboring routers. MUCA simply notifies the neighbor router that a routing update is available, and expects the neighbors to retrieve this incremental update. Thus, it effectively reduces the routing overheads.

We have conducted extensive simulations to demonstrate

ISBN 978-3-903176-08-9

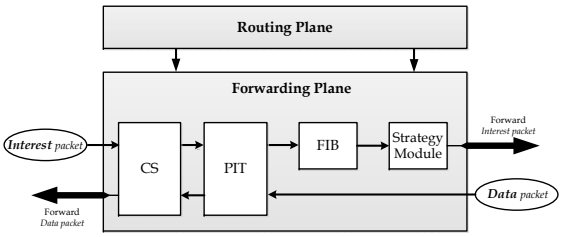


Fig. 1: Forwarding and routing planes in an NDN router

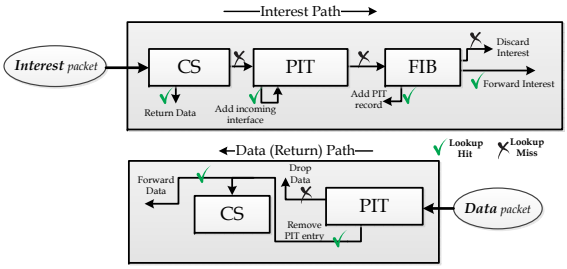


Fig. 2: Interest/Data processing in the forwarding plane

MUCA's benefits for multipath routing, in-network caching, and LSDB synchronization. Compared to the last version of NLSR, as the current *de facto* routing protocol of NDN testbed, MUCA enables multipath routing with 94% less traffic overhead, 26% faster content retrieval (by explicit support of in-network caching), 27% less overall cache space usage, and 22% less engaged routers for caching a specific content. At the same time, MUCA achieves fast reaction to failures due to quick routing update propagation and multipath support.

The remainder of this paper is organized as follows. Section II describes how routing and forwarding planes are decoupled in NDN and motivates our study. The design and operation of MUCA are detailed in Section III. The simulation results are presented in Section IV. Section V discusses the related work, and finally, Section VI concludes the paper.

II. Background and Motivation

A. Routing vs. Forwarding

IP forwarding plane is neither adaptive nor intelligent, as it strictly follows the routing plane [21]. NDN implies a substantial re-engineering of the forwarding plane and changes the role of routing plane from a directive to a consultant [10]. Thus, the routing plane is a second-class citizen in NDN. Actually, the routing plane only computes the route(s) towards each producer and provides the forwarding plane with this information. Instead, a forwarding strategy module is responsible for controlling all forwarding decisions (i.e., whether, where, and when to forward an Interest). Thus, unlike in IP, the forwarding table is not under control of the routing protocol and continuously updated according to forwarding plane performance measurements and administrative policies. This intelligent and adaptive forwarding plane enables exploring more radical and scalable routing schemes that are not possible in IP networks. Fig. 1 shows the forwarding and routing planes in an NDN router. It is worth noting that our main focus in this paper is on the routing plane. The way the paths are used by the strategy module depends on administrative decisions and the adopted forwarding strategies, which is beyond the scope of this paper.

In the NDN's request-driven (pull-based) communication model, a node requests a named content using an *Interest* packet. The intermediate neighbor nodes remember the interface from which this packet was received, and forward it by consulting their forwarding tables. Any node receiving the Interest packet and having the requested content simply responds with the corresponding *Data* packet. Unlike IP-based networks, not all of the packets need to be routed in NDN. Only the Interest packets are routed and the corresponding Data packets are returned based on the state information set up by the Interest packets in intermediate nodes (symmetric Interest-Data exchange).

As shown in Fig. 1, along with the strategy module, NDN has three main tables in its forwarding plane [23]: (1) CS (Content Store), a cache memory, that stores previously retrieved Data packets, (2) PIT (Pending Interest Table) that stores unsatisfied Interests as well as the interfaces through which they have been received, and (3) FIB (Forwarding Information Base) that serves as the forwarding table to direct the Interests towards the potential provider(s) of matching Data. Fig. 2 shows how Interest/Data packets are processed in the NDN tables in both sending and returning paths. Upon arrival of an Interest packet at a router, CS is searched for the requested name. If the desired content is found, then a Data packet is returned else PIT is searched. If the name matches a PIT entry, meaning that an Interest for this data has already been forwarded upstream, we just add the incoming interface to the related entry in PIT and wait for the response Data. Otherwise, after assigning a new entry to the requested name in PIT, the Interest is forwarded to the next hop(s) based on the forwarding strategy looking at FIB. If multiple next-hops exist in a FIB entry, the forwarding strategy determines how to use the multiple routes for forwarding Interests. On the return path, if the desired content arrives after its expiration time, it will be discarded; otherwise, after caching the content in CS, it will be sent through the interfaces listed in the matching PIT entry.

In-network caching is a gift from CS. Thanks to this built-in opportunity, all the NDN routers can act as temporary providers, thus unneeding to traverse the entire network for a desired content. Multipath support is a gift from PIT. Actually, PIT makes the NDN forwarding plane stateful, thus ensuring loop-free forwarding. This brings the opportunity of sending an Interest out through multiple interfaces in each router.

B. Motivation

In-network caching and multipath forwarding support are two prominent features of NDN. However, state-of-the-art studies still focus on the traditional problem of forwarding content requests to content producers, without explicit or efficient support of these two built-in opportunities in NDN,

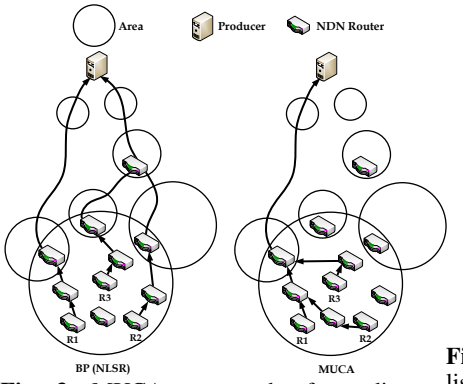


Fig. 3: MUCA merges the forwarding paths for the same content as early as possible (in the worst case, at the same border router). (The areas simply follow the network partitioning in OSPF.)

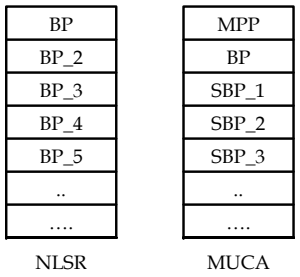


Fig. 4: MUCA vs. NLSR: a ranked list of routes (BP (Best Path), SBP (Semi Best Path), and MPP (Most Probable Path))

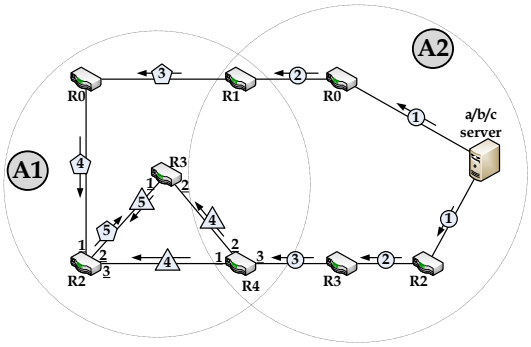


Fig. 5: Update dissemination after adding the `/a/b/c` server

which will limit the NDN's potential and benefits to applications. To alleviate this deficiency, while also improving scalability in terms of both computational and traffic overheads, we propose a new routing protocol which (1) mimics cache-awareness to the routing plane, and (2) computes multiple paths efficiently.

Caching: For any satisfied Interest, the response is cached in all nodes on its returning path to the requester. The story of routing in NDN is deficient without caring the caching capability in the routing plane. Simply enjoying this capability in the forwarding plane without explicitly exploiting in-network caching, as NLSR does, can degrade the performance in terms of both content retrieval delay and traffic (Interest/Data) overheads. Fig. 3 shows a simple example, where three nodes (R1, R2, and R3) request the same content. Ignoring the in-network caching capability in the state-of-the-art—which relies on the shortest paths—can cause forwarding the Interests through late, even never, merging paths (R3&R2 and R1&R2, respectively). Our idea is to label incoming route announcements and updates (advertised in the case of both topology and name prefix changes) at border routers such that all the internal routers will select the same border router for the same name prefix. This not only guarantees that requests for the same content will always merge before going out of the network (see R1&R2&R3's paths), but also improves the chance that they merge even before reaching the border router (see R1&R2's paths). Thus, MUCA equips the forwarding plane with a new path (referred to as *MPP*), explicitly exploiting in-network caching in NDN.

Multipath: Traditional routing protocols only compute a single best path. To implement multipath routing, NLSR provides a ranked list of all possible paths towards each producer. In this line, as a Link State (LS) routing protocol, it has no way except to run the Dijkstra algorithm from every single interface's point of view. This approach incurs high computational overhead, especially for routers with high connectivity. To address this issue, our general idea is to run Dijkstra only once in each router—this surely provides the best path (referred to as *BP*) towards each producer—and then ask for help from neighbors using their precomputed paths. This way, MUCA borrows a Distance Vector (DV) mechanism, realizing a cooperative routing for NDN. Thus, MUCA provides a list of alternative paths (referred to as SBPs) for each BP in much lower complexity in the price of a little more communication overhead.

In conclusion, as shown in Fig. 4, MUCA provides the forwarding plane with a new list of forwarding paths—including MPP, BP, and a dynamic list of SBPs as discussed later—while effectively utilizing caching and multipath in the routing plane.

III. MUCA Design

This section describes our design and its essential parts—multipath routing and LSDB synchronization. We simply follow the network partitioning in OSPF [1] and use a general hierarchical naming model, where each router takes a unique name `/<1-st Area ID>/<2-nd Area ID>/.../<n-th Area ID>/<Router ID>` in ascending order of the areas in which it resides. For example, in Fig. 5, internal router R0 in A1 and border router R4 in both A1 and A2 are simply named `1/0` and `1/2/4`, respectively.

A. Multipath Routing

MUCA comes with three different characterized paths: (1) Best Path (BP), provided by Link State (LS) face; (2) Semi Best Path(s) (SBPs), provided by Distance Vector (DV) face; and (3) Most Probable Path (MPP), provided by effectively exploiting the caching opportunity built in NDN. As the only characteristic in common, all of these three paths route the packets towards the content producers (i.e., original providers). While BP and SBPs ignore the in-network caching capability, MPP tries to meet the network caches as early as possible on its way to the producer.

To describe different paths, Fig. 5 shows an example in which a new `/a/b/c` server announces the content it serves (i.e., `/a/b/c`) by disseminating a routing update in the network. Each sequential step of dissemination is shown as a number next to each link. The shape changes to pentagon and

triangle when the update packet passes border routers `1/2/1` and `1/2/4`, respectively.

1) Scalable Multipath Support:

BP: It is the shortest path between every pair of routers, and used to deliver an Interest packet to its associated producer(s) with a minimum cost. BPs are calculated by the LS face, which runs Dijkstra algorithm in each router, having the global information of its area(s).

After synchronizing LSDBs in an area (see Section III.B), the MUCA's LS face determines BPs and builds the routing table (a.k.a. Routing Information Base (RIB)) for each router to all the others in that area. (For a router, its routing table, RIB, is populated with the costs to reach the other routers within its area.) Fig. 6 shows the RIBs of routers R2, R3, and R4 in area A1 in Fig. 5 after determining BPs, where all link costs are assumed to be 1 for simplicity. To differentiate the interfaces of a router, there is a number next to each link (as also shown in Fig. 5). *BP cost* and *BP interface* at each router are associated with the shortest-path cost and the interface through which the path reaches another router within the same area. For example, *BP interface* and *BP cost* in R3's RIB towards R0 in A1 are 1 and 2, respectively.

SBP: We propose using SBP as a supporting/alternative path for BP. We exploit the potential for cooperation between neighbors offered from the DV face of MUCA. Although DV is unsuitable for large wide-area networks, we borrow its main concept (i.e., querying only the neighbors) to use their "pre-computed" routing tables for finding the Semi Best Paths—called SBPs. It is worth noting that having the complete routing information of neighbors in a router, the number of SBPs towards a provider can be dynamically adjusted according to the administrative policies. Thus, unlike for BP (and MPP), MUCA can provide the forwarding engine with a ranked list of SBPs. However, for simplicity, we only present it as a single path in this paper. SBP can be exploited not only as a backup path for fault-tolerant routing but also for other purposes, such as load-balancing. However, how the strategy module uses this path is beyond the scope of this paper which focuses on the routing plane.

After determining BPs, the DV face resolves SBPs for each router to all the others within the area. To this end, each router sends a query to all of its neighbors and asks for their RIBs. This query is an Interest packet with name `</InterestSBP>`, and each router replies with its RIB (which includes its BP to each destination Y). Then, the SBP to Y passes through the BP of a directly connected neighbor providing the minimum cost. Note that SBP from a router X to a router Y needs to satisfy two conditions to ensure a loop-free routing: (1) SBP cannot go through the same X's interface as BP towards Y, and (2) BP to Y from the neighbor cannot cross X again. Fig. 6 shows the packets exchanged to resolve SBPs for R3. For example, as Figs. 6 and 7 show, R3 chooses R2 as its SBP next-hop to R4, while satisfying two conditions (1) BP and SBP to R4 go through different interfaces, and (2) R3 is not on R4's BP to R2. Moreover, the *SBP interface* and *SBP cost* in R3's RIB towards destination R4 are 1 and 2, respectively, since R2's BP to R4 has cost 1 and R3 reaches R2 through interface 1 just in one hop.

One may argue that as BP and SBP from a source router are not necessarily disjoint paths, our approach is not fault-tolerant enough. However, note that, unlike IP's end-to-end packet delivery model, the NDN forwarding policies are applied in a hop-by-hop manner. Thus, in the case of a link/node failure along the BP, its immediate neighbor quickly replaces the forwarding path by SBP (i.e., a part of BP is replaced by another path towards the producer). From the source router's point of view, the current forwarding path is not the best until it becomes aware of the failure and updates its RIB and FIB. Thus, the synchronization time (called the *convergence time* for fault-tolerant routing) is key to the performance of a multipath routing (see Section III.B).

Finally, note that SBP is equal to the NLSR's second best path (except in case that the second best path goes through the same interface as BP). Thus, it offers almost the same performance by incurring much lower computational overhead.

2) In-network Caching Support:

MPP: As mentioned earlier, MUCA also equips the forwarding plane with another path—called *Most Probable Path* (MPP)—through which an Interest will likely meet the desired content before reaching the producer(s). By using MPPs, we try to forward similar traffic (Interests) via the same (even partly) path to maximize utilizing built-in caching in NDN. To this end, we send all the requests which target the same name prefix through the same border router. We consider the case where routing announcements/updates are received in an area through multiple border routers. Then, the entry points (i.e., border routers) simply update a field *Modified Time* in receiving announcements/updates—this is referred to as *labeling process*. Finally, the internal routers choose the border router informed of a new name prefix before the others (i.e., that with the least *Modified Time*) and use their BPs (and SBPs) towards this border router to forward their similar requests. The same scenario applies sequentially in other areas. Finally, from a given router's point of view, there is a path (maybe longer than BP) which eventually reaches the producer(s), but with a higher probability of satisfying its request by an intermediate router.

For example, from the vantage point of router R3 in A1 in Fig. 5, its MPP to `/a/b/c` server goes through router `1/2/1` (R1). This is because the advertisement of this name prefix has been received by router `1/2/1` at the second hop, and propagated into the network (follow pentagon-shaped updates) before router `1/2/4` (follow triangle-shaped update) receiving the update at the third hop. Finally, BP and MPP towards `/a/b/c` server are [`1/3-1/2/4-2/3-2/2`] and [`1/3-1/2-1/0-1/2/1-2/0`], respectively. Note that the resolved MPP in a router towards `/a/b/c` server may be the same as BP or SBP (e.g., for routers `1/0` and `1/2`), or different (as described for router `1/3`). Thus, although MPPs may take longer paths than BPs, they can effectively reduce content retrieval delay. This is because MPPs from all internal routers for `/a/b/c` are directed to the same border router in

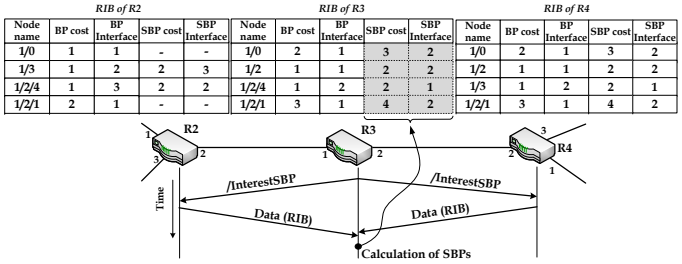


Fig. 6: RIB (routing table) update and SBP calculation by exploiting the distance-vector face of MUCA

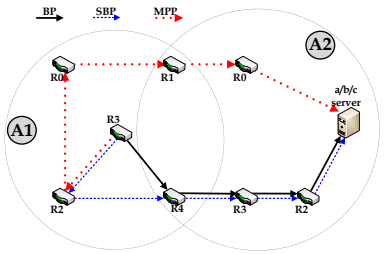


Fig. 7: BP, SBP, and MPP from R3 to retrieve `/a/b/c`

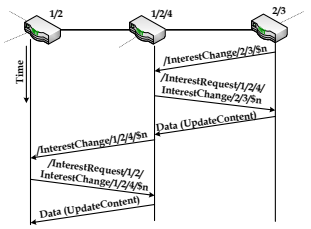


Fig. 8: Notifying the neighbor routers to synchronize their LSDBs

an area, which not only guarantees that the paths merge before going out of the area, but also improves their chance to merge even before reaching the border router.

To realize MPP, we define MUCA header, including three fields *Area ID*, *Border Router*, and *Modified Time* augmented with NDN Data header. These fields experience no change in the return path while passing internal routers. However, upon arrival at a border router, they are updated according to the current area, border router, and arrival time.

By presenting MPP, we prevent scattered caching and forwarding Interests through improper paths or towards deprecated copies of contents. Moreover, by avoiding sending several similar Interests throughout the network, MPPs reinforce the role of PIT, as one of the NDN main design principles. This way, we save more network bandwidth and decrease the possibility of congesting the intermediate links/routers, and reduce the transport cost incurred by traffic between networks. By receiving fewer Interest packets at the producers, the servers' load will also be reduced.

Note that MUCA can support multipath routing not only for a single content producer by leveraging BP, SBP, and MPP (Fig. 7 depicts BP, SBP, and MPP from router `1/3` to `/a/b/c` server), but also for multiple producers (in case there are more `/a/b/c` servers in the network).

B. LSDB Synchronization

As part of any LS routing protocol, to synchronize all the LSDBs (Link State Databases), each router needs to detect a new update in the case of both topology and name prefix changes and disseminate it throughout the network. (The LSDB at each router contains information on reachability to both routers and name prefixes.) In this line, the latest version of NLSR [17] uses ChronoSync [25]. Considering the routing update propagation problem as a synchronization problem, two neighbors need to periodically inform each other about the state of their LSDBs (even if there is no changes in the network). Although there are some benefits with ChronoSync, especially in highly dynamic and unreliable scenarios, it is not a perfect fit to NDN routing protocol synchronization, so incurring high message overhead. To address this problem, MUCA suggests that each router simply notifies its neighbors that a routing update is available, and expects the neighbors to retrieve this incremental update. This mimics a push-based notification of the exact changes in heart of the NDN pull-based communication model, which effectively reduces not only the convergence time but also many unnecessary periodic control packets to detect the updates and difference between LSDBs, if any.

Implementing LSDB Synchronization: The currently adopted update model in NDN is request-driven, requiring all the routers to pull the updates. To implement our approach, in the case of any update in a node, it sends an *InterestChange* to its neighbors and they respond by returning *InterestRequest*s to pull new changes (as a Data packet *UpdateContent*). Upon receiving the changes at a neighbor, it updates its LSDB and notifies its neighbors. This procedure is repeated hop-by-hop until all the routers in the associated area are informed of any update. Names `/InterestChange/<origin router name>/<nonce>` and `/InterestRequest/<sender router name>/InterestChange/<origin router name>/<nonce>` are used for InterestChange and InterestRequest packets, respectively, where the suffix of InterestRequest is identical to its associated InterestChange. Here, *origin router name* and *sender router name* refer to the routers from which InterestChange and InterestRequest are transmitted, respectively, and *Nonce* is a random integer. Fig. 8 shows LSDB synchronization process for routers `2/3`, `1/2/4`, and `1/2` in Fig. 5, where $\$n$ is a *Nonce*. After receiving the routing update at router `1/2/4` from router `2/3`, it updates its LSDB and then sends InterestChange to its neighbor (i.e., router `1/2`). Upon receiving this packet, router `1/2` requests an update by returning an InterestRequest. Finally, router `1/2/4` generates an UpdateContent by which router `1/2` can update its LSDB and inform its neighbor of this new change.

IV. Evaluation

In this section, we evaluate the performance of MUCA via extensive simulations using ndnSIM, which is the de facto simulator of NDN. The results are compared to the last version of NLSR [17], as the current de facto routing protocol of NDN testbed, to demonstrate MUCA's benefits for multipath routing, LSDB synchronization, and in-network caching utilization. The simulation parameters are set to their default values in ndnSIM [3]. The topology generator aSHIIP v.3 [2] and the GLP model [4] are also employed to generate random networks. The results are the average of 10 runs.

Scattered caching may waste and improperly leverage cache capacity.

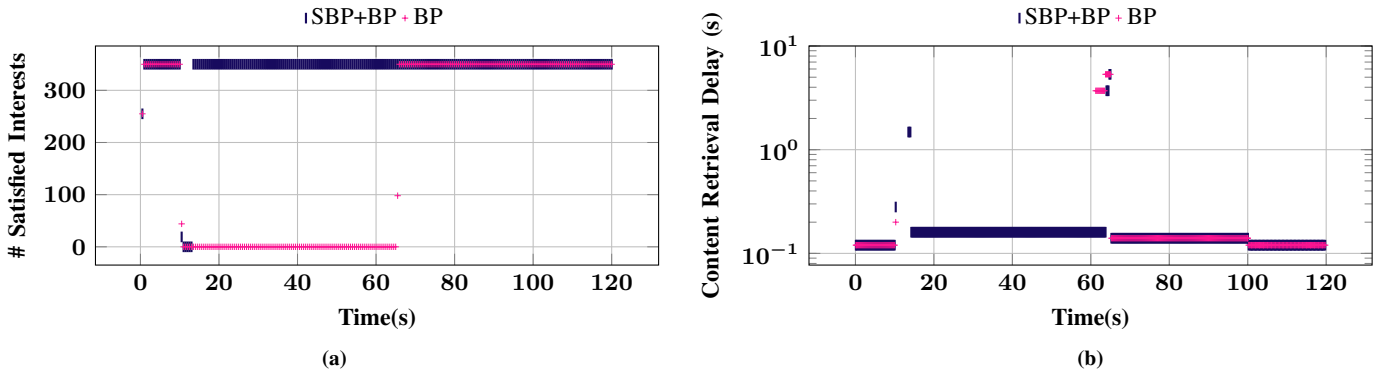


Fig. 9: The benefits of multipath routing in a node failure scenario: (a) number of satisfied Interests and (b) content retrieval delay

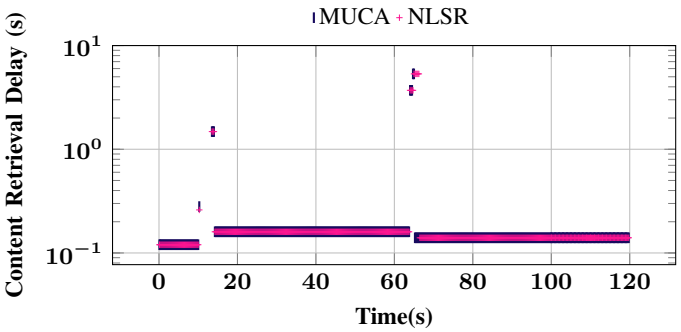


Fig. 10: Semi Best Path (SBP) in MUCA achieves the performance of second Best Path in NLSR with much lower routing overheads

A. Multipath Support

We answer three questions on the performance of our proposed multipath routing protocol exploiting LS and DV faces: (1) what are the benefits of employing multipath? (2) how MUCA and NLSR outperform each other during a failure scenario?, and (3) what is the cost of resolving SBPs in terms of traffic overhead?

To answer the first question, Fig. 9 compares two modes, BPs alone and BPs & SBPs together, in terms of number of satisfied Interests and consumer-perceived response time (a.k.a. content retrieval delay) in case of node failure. The retrieval delay is the time duration between sending an Interest and receiving its corresponding Data, including the time for retransmissions. We capture network events during a period of 120 seconds, where two randomly selected nodes act as the producer and consumer residing in two sides of a 100-node network partitioned into four areas. After computing and populating the LSDBs in all the routers, at the 10-th second, we brought down a node on the consumer's BP towards the producer. In MUCA with only a single path, the node failure triggers the Dijkstra algorithm to calculate the new BP and resolve the desired content. As shown in Fig. 9(a) and (b), it takes about 60 seconds to detect the node failure and then to converge (thus, using only BPs, no Data returns during this period). This figure clears the importance of using multipath, as using a single path (i.e., BP) drastically increases packet drop rate (see Fig. 9(a)) and content retrieval delay (see Fig. 9(b)), during convergence time. However, providing the forwarding plane with SBPs lets it forward Interests on another path almost immediately from the failed node's neighbor, until the network converges and the consumer updates its BP (around the 70-th second). Finally, at the 100-th second, when the failed node is recovered, the traffic is switched back on the old path. Fig. 9 thus illustrates the benefit of employing SBPs along with BP, and its vital effect on overall performance of the network.

To answer the second question, Fig. 10 compares MUCA with NLSR in the same scenario except that the failed node is not brought back up. Both MUCA and NLSR enable the forwarding plane to quickly switch to the alternative paths (SBP and the second best path, respectively) and continue transmission of packets with a larger retrieval delay. After detecting the node failure (around the 70-th second as mentioned earlier), both MUCA and NLSR daemons start propagating this change throughout the network and updating LSDB of the routers consequently. As evident from the figure, both protocols perform almost the same, though MUCA could converge a little faster due to its update propagation mechanism. After convergence time, both protocols switch to a new BP with a relatively longer delay than the initial one. In conclusion, using SBP, MUCA can achieve the performance of NLSR only by incurring little traffic overhead to the network, while drastically reducing the computational overhead of NLSR.

Finally, to answer the third question, Fig. 11 shows a complete analysis of traffic overhead of resolving SBPs. To give a broader view, we consider different sizes of the network, i.e., 50, 80, 100, and 150 nodes. (We attempted to cover medium to large networks (as the AT&T core network topology has 154 nodes [9].)) We also change parameter p in the GLP model to

This is long enough for warming up and testing the behavior of network in terms of convergence time.

As in OSPF, tracking the network connectivity is handled by sending Hello packets to neighbor nodes. Usually after hearing nothing from a neighbor within three continuous Hello packet interval, the neighbor is determined as dead.

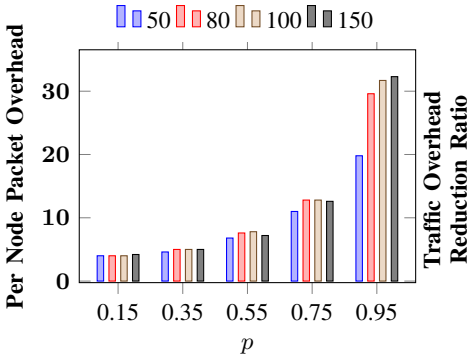


Fig. 11: Average "per-node" packet overhead to resolve SBPs for different network sizes (50, 80, 100, and 150 nodes) and node degrees (p)

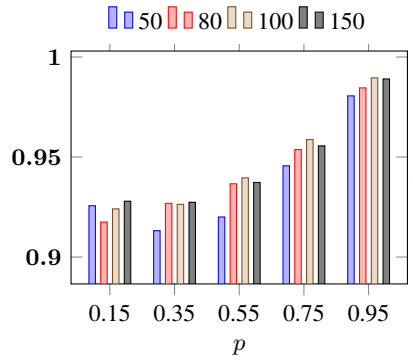


Fig. 12: MUCA vs. NLSR: MUCA effectively reduces traffic overhead by utilizing a new LSDB synchronization mechanism

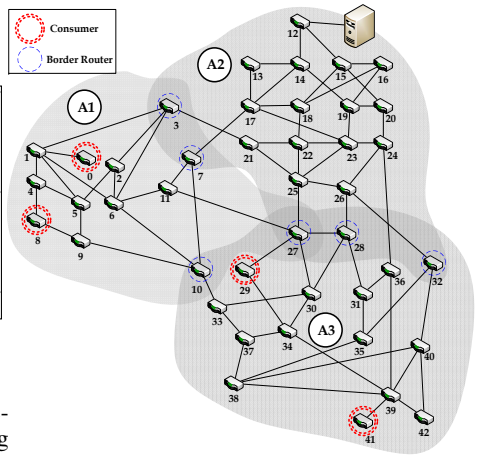


Fig. 13: Network topology to evaluate the MPP performance

create different node degrees, ranging from very sparse to very dense deployments, where p $(1-p)$ specifies the probability of adding a predefined number of new links (a new node) to the network at each time-step. The results show that this overhead is negligible. For example, when there are 80 routers in the network and the density is normal (i.e., $p = 0.5$, which means the network is neither fully mesh nor sparse), less than 10 packets per node is needed to resolve SBPs.

B. LSDB Synchronization

We now evaluate the performance of MUCA against NLSR in terms of synchronization overhead. In each scenario, a random router is informed of a new name prefix and updates its LSDB. Then, the number of transmitted packets (including all periodic and non-periodic ones) are measured. Fig. 12 shows the traffic overhead reduction ratio provided by MUCA to synchronize all the LSDBs in the network. MUCA is shown to significantly outperform NLSR, especially in larger networks. Moreover, by increasing p, the number of links grows, so more periodic packets will be exchanged by NLSR. This figure casts doubt on using ChronoSync (or similar algorithms) as it performs poorly compared with a straightforward approach. Instead, by blocking many unnecessary packets (e.g., those exchanged for finding differences between LSDBs), our approach reduces the traffic overhead on average by 94% in the network.

C. In-Network Caching Support

As mentioned earlier and also shown in Fig. 4, MUCA provides the forwarding plane with a new ranked list of the candidate paths, giving priority to MPPs. In this section, we compare the NDN forwarding based on MPP versus BP (NLSR approach) to see whether MUCA's ranking rubric outperforms NLSR. Moreover, to fully demonstrate the pros and cons of MPP-based forwarding, we also compare it with "flooding" (as the simplest de facto forwarding strategy in NDN). All three schemes follow regular caching of Data packets at intermediate nodes and use LFU as their replacement policy [14]. The performance of in-network caching is evaluated in terms of: (i) content retrieval delay, (ii) overall cache memory usage, and (iii) number of nodes engaged in caching.

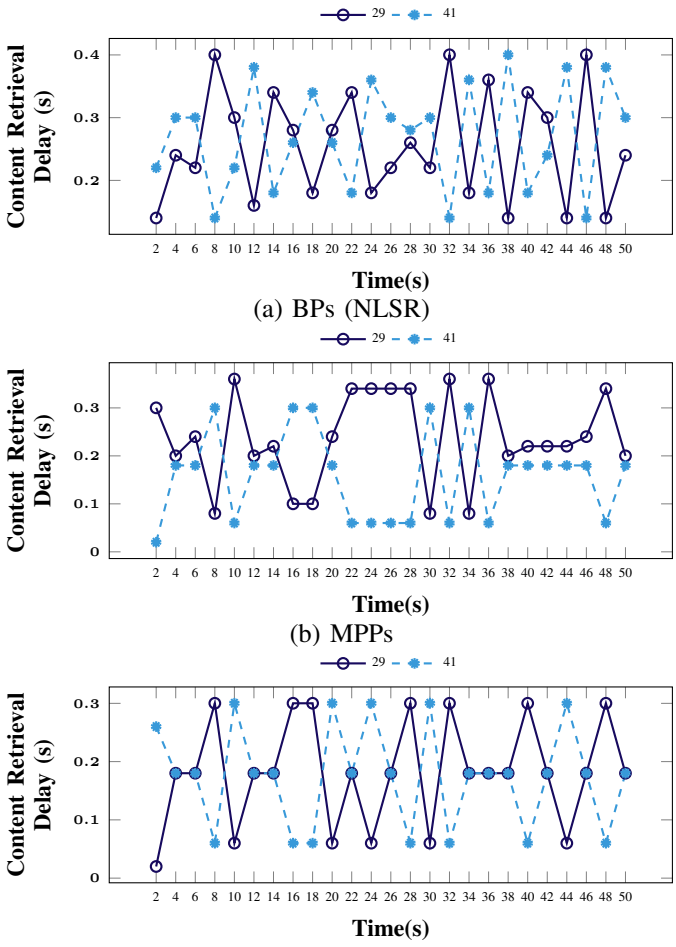


Fig. 14: Content retrieval delay in nodes 29 and 41 using different forwarding schemes

Fig. 13 illustrates a network of 44 nodes partitioned into

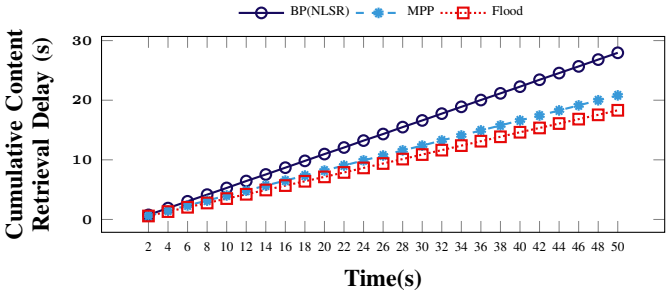


Fig. 15: Cumulative content retrieval delay using different forwarding schemes

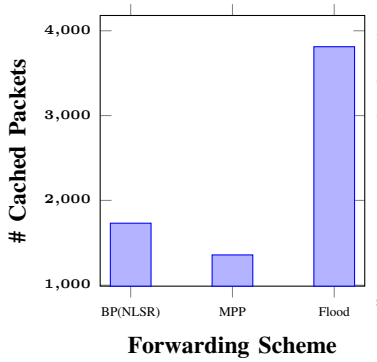


Fig. 16: Number of packets cached in the network using different forwarding schemes

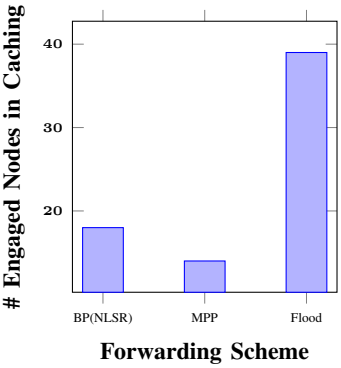


Fig. 17: Number of nodes engaged in caching using different forwarding schemes

three areas, where four consumers request 20 name prefixes served by one server (at the top). The location of each consumer is also denoted by double-dotted circles. In order to test the network operation, each time step is randomly chosen between 0 and 4 seconds, in which each of four consumers requests one of the prefixes. At the end of simulation that lasts for 50 seconds, each node has requested at least a half of the existing name prefixes. Thus, it is highly possible that a content is repeatedly requested by different consumers (time-locality principle). Although each content in CS is valid only for a short while in reality, we enjoy the caching opportunity built in NDN.

Fig. 14(a)-(c) shows the content retrieval delay for nodes 29 and 41 (two of four consumers) in A3 for all three forwarding schemes. This delay for the nodes under the MPP-based scheme is shown to be higher than that under the flooding scheme, while smaller than that under the BP-based one. Indeed, although leveraging only BPs can result in the same or even better performance under a special condition (when the content requisition does not follow the time locality principle), MPPs can, in general, reduce the retrieval delay. As evident from the figure, nodes 29 and 41 have relatively opposite retrieval delay trends (i.e., high vs. low). This verifies that by leveraging in-network caching, if a node needs a content which has already been requested, it will meet a cached version with high probability. For example, as evident from the 22-nd till the 28-th second in Fig. 14(b), node 41 retrieves a desired content very quickly as it has been already consumed by node 29. For the majority of time, the content can be retrieved in less than 0.2 second in Figs. 14(b) and (c), while this is reversed in Fig. 14(a).

Fig. 15 illustrates the cumulative retrieval delay for all consumers using different forwarding schemes. As evident from the figure, the MPP-based forwarding scheme performs close to flooding which is our lower bound—flooding minimizes the delay in the cost of incurring the maximum traffic overheads to the network. Actually, the MPP-based scheme outperforms the BP-based one on average by 26% in this scenario, while flooding reduces the retrieval delay on average by 34% and 12% compared to BP- and MPP-based schemes, respectively.

Fig. 16 shows the cumulative cache memory usage by each scheme in terms of the number of cached packets at the network nodes. As evident from the figure, the MPP-based scheme decreases the overall CS space usage on average by 27% and 64% compared with BP-based and flooding schemes, respectively. Besides, as shown in Fig. 17, forwarding the Interests over MPPs can reduce the number of nodes engaged in caching a specific content on average by 22% and 64% compared to BP-based and flooding schemes. Obviously, fewer engaged nodes mean less scattered caching, while also providing a lower retrieval delay by using MPP compared to the BP-based scheme. In general, scattering the content between more network nodes (as flooding does) results in a greater opportunity to reduce the retrieval delay. However, we did not follow this to avoid caching redundant contents. Finally, the MPP-based scheme consumes CS space very efficiently while also performing very close to flooding. Based on these observations, we can make a trade-off between the content retrieval delay and the CS space usage by choosing MPPs or flooding.

Knowing the benefits of using MPP over BP in different aspects, we can reject adopting the assumption of several studies (like NLSR) which imply using path cost as a metric to rank the available paths. Instead, we believe that the paths with higher probability to meet the cached content have higher priority over the shortest paths towards the provider(s) in NDN. That is why MUCA gives MPP a higher priority than BP and SBPs in its ranked list of paths as shown in Fig. 4.

V. Related Work

There exists several studies on routing in NDN. OSPFN [18], as an extended version of OSPF [1], is the first NDN routing protocol for rapid prototyping of name-based forwarding in the NDN testbed. However, it suffers from several critical drawbacks such as IP dependency, employing GRE tunnels, and disregarding multipath forwarding. The two-layer routing protocol in [6] uses OSPF to resolve topology and calculate the shortest-spanning trees. However, it relies on flooding for update dissemination and does not yet support multipath routing towards a single producer. To mitigate these problems and allow for new topology-discovery methods, a named-data link state routing protocol (NLSR) was proposed in [10], [17]. However, it—as the current de facto routing protocol of NDN

testbed—suffers from high computational and traffic overheads. A controller-based routing scheme (CRoS) for NDN was also proposed in [15], which uses multiple controllers to achieve scalability. However, it incurs high traffic overhead due to flooding of Interests to search for controllers. LSCR [9] proposed a name-based link-state routing protocol which aims to provide forwarding plane with permanent loop-free paths. DCR [8] is the first name-based content routing which does not require any information about physical topology and works solely based on distance information. However, both LSCR and DCR refuse to provide the network with information of all available providers, while none of them explicitly employ in-network caching capability, as well. Bloom Filters (BF) are used in [5], [11], [12], [20] to digest FIB and exchange information about content availability, but they incur high signaling overheads. The stateful BF is also used in [16], but it only leverages the passive mode of prefix announcements, thus flooding the network multiple times. Although BF can reduce the space complexity, it suffers from false positives (collisions) which, in turn, degrade performance. Besides, Wang *et al.* [19] and Zhang *et al.* [24] attempted to utilize in-network caching by adding new data structures or modifying the existing ones. However, they preserve the relationship with IP-based routing, which does not meet the NDN's goal of departing from IP [10].

There have also been other efforts [7], [13] focusing on "inter-domain" routing and leveraging the concept of BGP which are beyond the scope of this paper.

VI. Conclusion

Explicit support of in-network caching and multipath forwarding from routing protocol is key to realize NDN. This paper proposed MUCA as a stand-alone intra-domain routing protocol for NDN and highlighted its important features. It makes three main contributions: (1) a combination of link-state and distance-vector routing protocol classes to efficiently support multipath routing, (2) a new path, different from the regular forwarding paths, to effectively exploit built-in caching opportunity in NDN, and (3) a new mechanism for LSDB synchronization, where the incremental routing updates are simply notified to the neighbor routers. Finally, MUCA equips the forwarding plane with a new ranked list of forwarding paths. Our in-depth evaluation demonstrates the benefits of MUCA and its superiority over NLSR, the current de facto routing protocol of NDN testbed.

We expect that MUCA will play a key role in NDN, as what OSPF has done in IP-based networks. MUCA can easily accommodate new ideas/solutions thanks to its flexible design. MUCA is, therefore, a good starting point towards a comprehensive routing solution for NDN.

Acknowledgments

This work was supported in part by NSF under Grants CNS-1345142 and CNS-1629009.

NDN can work in two name-announcement modes (i) active mode (where the prefixes are announced by producers), and (ii) passive mode (where the prefixes are solicited by consumers).

References

[1] "OSPF Version 2," https://www.ietf.org/rfc/rfc2328.txt, [Online].
[2] "Supelec," http://wwwdi.supelec.fr/software-orig/ashiip/, [Online].
[3] A. Afanasyev *et al.*, "ndnSIM: NDN simulator for NS-3," Tech. Rep. NDN-0005, 2012.
[4] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," in *IEEE INFOCOM'02*, 2002, pp. 638–647.
[5] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "INFORM: A dynamic interest forwarding mechanism for information centric networking," in *ICN'13*, 2013, pp. 9–14.
[6] H. Dai, J. Lu, Y. Wang, and B. Liu, "A two-layer intra-domain routing scheme for named data networking," in *IEEE GLOBECOM'12*, 2012, pp. 2815–2820.
[7] S. DiBenedetto, C. Papadopoulos, and D. Massey, "Routing policies in named data networking," in *ACM SIGCOMM workshop on Information-centric networking (ICN'11)*, 2011, pp. 38–43.
[8] J. Garcia-Luna-Aceves, "Name-based content routing in information centric networks using distance information," in *ACM Conference on Information-Centric Networking (ICN'14)*, 2014, pp. 7–16.
[9] E. Hemmati and J. Garcia-Luna-Aceves, "A new approach to name-based link-state routing for information-centric networks," in *ACM Conference on Information-Centric Networking (ICN'15)*, 2015, pp. 29–38.
[10] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "NLSR: Named-data link state routing protocol," in *ACM SIGCOMM workshop on Information-centric networking (ICN'13)*, 2013, pp. 15–20.
[11] M. Lee, K. Cho, K. Park, T. Kwon, and Y. Choi, "SCAN: Scalable content routing for content-aware networking," in *IEEE ICC'11*, 2011, pp. 1–5.
[12] H. Liu, X. De Foy, and D. Zhang, "A multi-level DHT routing framework with aggregation," in *ACM SIGCOMM workshop on Information-centric networking (ICN'12)*, 2012, pp. 43–48.
[13] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented networks," in *ACM CoNEXT'08*, 2008, pp. 1–6.
[14] S. Tarnoi, K. Suksomboon, W. Kumwilaisak, and Y. Ji, "Cooperative routing protocol for content-centric networking," in *IEEE LCN'13*, 2013, pp. 699–702.
[15] J. V. Torres, L. H. G. Ferraz, and O. C. M. B. Duarte, "Controller-based routing scheme for named data network," Electrical Engineering Program, COPPE/UFRJ, Tech. Rep., 2012.
[16] M. Tortelli, L. A. Grieco, G. Boggia, and K. Pentikousis, "COBRA: Lean intra-domain routing in NDN," in *IEEE CCNC'14*, 2014.
[17] Y. Y. L. W. B. Z. V. Lehman, A. M. Hoque and L. Zhang, "A secure link state routing protocol for NDN," Tech. Rep. NDN-0037, Jan. 2016.
[18] L. Wang, A. K. M. M. Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFN: An OSPF based routing protocol for named data networking," University of Memphis and University of Arizona, Tech. Rep., 2012.
[19] S. Wang, J. Bi, and J. Wu, "Collaborative caching based on hash-routing for information-centric networking," in *SIGCOMM'13*, 2013, pp. 535–536.
[20] Y. Wang, K. Lee, B. Venkataraman, R. Shamanna, I. Rhee, and S. Yang, "Advertising cached contents in the control plane: Necessity and feasibility," in *IEEE INFOCOM'12*, 2012, pp. 286–291.
[21] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," in *ACM Conference on Information-Centric Networking (ICN'14)*, 2014, pp. 27–36.
[22] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, 2014.
[23] L. Zhang *et al.*, "Named Data Networking (NDN) Project," Tech. Rep. NDN-0001, 2010.
[24] X. Zhang, T. Niu, F. Lao, and Z. Guo, "Topology-aware content-centric networking," in *SIGCOMM'13*, 2013, pp. 559–560.
[25] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in named data networking," in *IEEE ICNP'13*, 2013, pp. 1–10.

Association Optimization in Wi-Fi Networks based on the Channel Busy Time Estimation

Mohammed Amer
Univ Lyon, UCB Lyon1, Inria,
ENS de Lyon, CNRS, LIP UMR 5668
Email: mohammed.amer@ens-lyon.fr

Anthony Busson
Univ Lyon, UCB Lyon1, Inria,
ENS de Lyon, CNRS, LIP UMR 5668
Email: anthony.busson@ens-lyon.fr

Isabelle Guérin Lassous
Univ Lyon, UCB Lyon1, Inria,
ENS de Lyon, CNRS, LIP UMR 5668
Email: isabelle.guerin-lassous@ens-lyon.fr

***Abstract*—With the centralized management paradigm offered by the recent IEEE 802.11 products, it is now easier and more efficient to optimize associations between access points (APs) and stations. Most of the optimization approaches consider a saturated network. Even if such traffic conditions are rare, the optimization of the association step under this assumption has the benefit to fairly share the bandwidth between stations. Nevertheless, traffic demands may be very different from one station to another and it may be more useful to optimize associations according to the stations' demands. In this paper, we propose an optimization of the association step based on the stations' throughputs and the channel busy time fraction (BTF). The latter is defined as the proportion of time the channel is sensed busy by an AP. Associations are optimized in order to minimize the greatest BTF in the network. This original approach allows the Wi-Fi manager/controller to unload the most congested AP, increase the throughput for most of the stations, and offer more bandwidth to stations that need it. We present a local search technique that finds local optima to this optimization problem. This heuristic relies on an analytical model that predicts BTF for any configuration. The model is based on a Markov network and a Wi-Fi conflict graph. NS-3 simulations including a large set of scenarios highlight the benefits of our approach and its ability to improve the performance in congested and non-congested Wi-Fi networks.**

I. Introduction

Wireless LANs can offer the possibility to mobile devices to access the Internet. In particular, due to its efficiency and facility of deployment, IEEE 802.11 (referred as Wi-Fi hereafter) has become a very popular wireless technology [1]. The density of access points (APs) allows the users to experience a high throughput and to be mobile without significant degradation of the link quality or connection interruption. Nevertheless, the limited number of non-overlapping channels makes difficult to ensure a good quality of service to users in dense WLANs without a dynamic and rational management. The management functions include channel assignment to access points, transmission power control, association between stations and APs, handovers, etc. In this work, we focus on the association. In Wi-Fi networks, association is the first step that allows a station to connect to the network. A station associates with an AP within its transmission range. If several APs are available in this area, the station will associate, generally, to the AP with the best RSSI (Radio Signal Strength Indicator). This metric, that measures the link quality between APs and stations, does not take into account the number of already associated stations neither the traffic load on the APs. It may lead to a heterogeneous distribution of stations among APs and consequently a bad distribution of the load in the network. Resources are therefore not optimally utilized, penalizing the overall performance of the network.

ISBN 978-3-903176-08-9

This problem and the need to facilitate AP administration have led to a centralization of the management in Wi-Fi networks [2], [3]. A Wi-Fi controller is in charge of all operations in the Wi-Fi network. It has a global vision of the network that enables a simple, flexible and efficient management of the resources. In particular, associations optimizing the resource usage can be computed in real time. The controller has then the ability to move stations from an AP to another using, for instance, the BSS transition management frames defined in IEEE 802.11v [4].

Most of the existing solutions to optimize associations propose solutions that aim to improve the overall network throughput and/or the fairness between stations in a saturated scenario [5]–[7]. The saturated scenario corresponds to a case where devices (stations and/or APs) have always a frame to send. This assumption is unrealistic but allows to express the minimum amount of throughput a device can obtain. Nevertheless, it does not take into account real traffic demands and the proposed association may be inaccurate. For example, if a part of the stations have very low traffic, the unused bandwidth can be reused by other stations with higher demands. A fair distribution of the resources may then be counter-productive: low traffic stations may be associated to the same AP which may be consequently idle whereas stations requiring high throughput are associated to overloaded APs.

In this paper, we propose an association optimization based on traffic demands. These traffic demands are defined as the downlink traffic from APs to stations and are measured in real time. The load of an AP is estimated through the **busy time fraction** (BTF). BTF corresponds to the fraction of time an AP senses the medium/channel busy due to its own transmissions or the ones from the other APs in its sensing range. This quantity is easily collected from the local Wi-Fi card statistics obtained on the current configuration. From a protocol point of view, it can be collected from the channel load request/report defined in IEEE 802.11k amendment [8]. In order to forecast its values for other configurations, we propose an analytical

model that estimates BTF. It is based on a Markov network, a conflict graph and the current traffic demands. Our optimization problem consists in finding the association that minimizes the greatest BTF in the network. This original approach allows the Wi-Fi controller to unload the most congested AP and offer more bandwidth to stations that needs it. Our solution is evaluated with NS-3 simulations that cover a large set of scenarios (ISM and UNII bands, existing topologies of Wi-Fi networks and random ones, different number of stations and input rates, TCP and UDP flows, etc.). The obtained results highlight the benefit of our approach and its ability to improve performance in congested and non-congested networks.

The paper is organized as follows. In Section II, we present the related work and our own contributions. The network model is described in Section III-A. The BTF is defined in Section III-B. The model estimating its value for any configuration is presented in the same section. Section IV introduces the optimization problem and the heuristic used to propose approximate solutions. Numerical results are shown and discussed in Section V. We conclude in Section VI.

II. Related work

The densification of Wi-Fi networks and the turn to its centralized management have motivated researches to optimize Wi-Fi configurations. Association between stations and APs is one of the key elements to improve the network performance. Below, we briefly summarize studies that address AP association. Contributions are classified as distributed, centralized and on-line.

Several approaches have proposed a distributed strategy. For instance in [9], the problem of AP association in WLAN is formulated through a mixed strategic game with a utility function that maximizes the throughput. They propose to users to move from their positions to improve their throughput. Distances traveled to a new AP are incorporated as a cost in the strategy game. The authors in [10] propose a solution for differentiated access service selection based on network applications, which are classified into four types according to their QoS requirements. Their approach can be used in a periodic or aperiodic strategy. In [11] a utility-based strategy is proposed to select the best AP according to the distance, data rate and delay. These three metrics are normalized between zero and one. Then, an equal weight is given to each metric within the utility function. The AP with the highest utility value is selected. Authors of [12] propose an algorithm that evaluates applications used by the stations, classified as data or voice, and changes the association accordingly. To achieve load balancing and good voice quality, the number of nodes connected to an AP and its RSSI are also considered in the association algorithm. However, evaluation is performed through a simulation of the model and not with a realistic network simulator. Moreover, one single data rate is considered for all transmissions between stations and APs. All these approaches assume only the saturated mode.

Centralized association has been proposed in order to achieve a global optimum. Wong et al. [13] propose a centralized max-min user throughput approach to optimize the AP re-association subject to a certain handover cost constraint. A multi-objective optimization function that maximizes the download user throughput and minimizes the number of handovers in saturated mode is also proposed in [14]. In [15], the authors formulate this problem as a non-cooperative game where each user tries to minimize its cost function, defined as the data transfer time. Their solution can be centralized or distributed. Authors of [16] propose a centralized approach to improve users' throughput in dense WLAN. They use signal-interference-noise-ratio (SINR) between APs and stations to control the association. In order to further coordinate interference and increase spatial reuse, an algorithm is proposed to adjust the clear channel assessment (CCA) threshold of the 802.11 MAC protocol. Taking into account the propagation environment, the authors of [17] investigate the impact of the AP deployments and station association in dense WLAN on the aggregate throughput.

On-line approaches have also been proposed in the literature. It consists in changing associations in real time, typically when an event occurs such as the arrival or departure of a station. In [18] the authors present a new AP selection metric. Their mechanism tries to maximize stations throughput as well as minimize its negative effect on high rate stations currently accommodated by the AP to which it wishes to associate. They propose two selection schemes based on this metric: a static one where stations only consider their association as well as a dynamic scheme where all associations are reevaluated from time to time. To improve the overall WLAN performance, Babul et al. propose in [19] an approach that considers simultaneously the channel assignment and the association control. However, validation is made through a simulation of the model and realistic Wi-Fi/network layers are not taken into account. Based on the Markov model to estimate the uplink and downlink throughput of clients, the authors of [20] propose an on-line AP association algorithm for 802.11n WLANs with heterogeneous clients. In this approach, authors seek to improve the overall network throughput.

All the cited approaches consider a saturated network except in [16] where the SINR is measured and used to determine data and error rates. Moreover, the traffic demand is not taken into account in the associations. The motivation of this paper is to design association algorithms able to adapt to traffic demands. It allows the controller to balance the load according to the real traffic, alleviate congested AP, and offer bandwidth to stations that need it. It is based on measurements available on most of the Wi-Fi products (e.g., busy time, data rates, error rates, etc.).

III. System Model

A. Network Model

We consider a general 802.11 WLAN consisting of a fixed number of APs. The set of APs is assumed to belong to the same extended service set (ESS) and is managed by a WLAN controller. We take into account only downlink traffic, from the APs to the stations, as downlink traffic is

preponderant compared to uplink traffic [21]. The controller is in charge of determining the association. When a new station connects to the ESS, it first associates with the default AP which is, for most of the implementation, the one with the best RSSI. The controller can, according to our algorithm, change associations at regular intervals or when a particular event occurs (arrival/departure of stations for instance). We assume that the controller collects periodically the following measurements from APs:

- the current association,
- the busy time fraction for each AP,
- the conflicts between APs (the conflict graph is formally defined later in this paper),
- for each station:
  - the data rates between APs and the station,
  - the throughput and the average frame size received by the station from its AP,
  - the error rate (or equivalently the probability of success) between the station and its AP.

It is worth noting that most of these measurements are already available on most of the AP products (e.g., Cisco Aironet Series APs).

When the controller finds out a better association, it triggers the corresponding changes: through control frames, stations can be disassociated from the current AP and associated to the new one. The application of a new configuration induces a re-association cost. The condition for applying a new association may be function of the cost and gain of the new configuration.

The association, proposed in this work, is based on the estimation of BTF. The BTF of the current configuration is known, but it has to be predicted for the other configurations that can be considered for a new association. Our prediction model relies on the following assumptions:

- **Data rate**: APs are able to determine the best data rate for all the stations in its transmission range (associated or not).
- **Throughput**: we assume that a station, associated to a new AP, will request at least the same throughput as in the current configuration.
- **Probability of success**: the probability of success for each station (probability that a frame is correctly received) is measured between APs and their associated stations. Its prediction for another association is difficult. In our model, we assume that this probability remains the same if the station does not change its channel when it reassociates. In case of a channel change, the success probability is set to the smallest probability of success among the stations already associated with the new AP.

The objective function based on BTF and the heuristic used to minimize it are presented in Section IV. We introduce, in the next section, the analytical model used to estimate BTF for all APs.

B. Busy Time Fraction estimation

BTF for an AP is defined as the fraction of time the channel is sensed occupied. This measurement can be obtained from the measurement reports of IEEE 802.11k or directly from the physical registers of the interface that measures the busy time according to the CCA mechanism [1]. This quantity is generally available for the current association. But in the context of our optimization, it is necessary to estimate this fraction for any other configuration.

In our model, we define b_j the busy time fraction for an AP j. This time is composed of two quantities: the local busy time fraction and the neighbor busy time fraction. The local busy time fraction, denoted b_j^L, corresponds to the time, per second, the channel is occupied by its own transmissions. This time takes into account the physical occupation of the channel and the access method times (back-off, DIFS, etc.). The neighbor busy time fraction, denoted b_j^N, is the proportion of time the channel is occupied by APs in its sensing range. It considers only the physical occupation of the channel corresponding to transmissions. We get,

$$b_j = b_j^L + b_j^N$$

1) Local Busy Time Fraction: This time includes the time to transmit data on the physical channel (T_{PHY}) to one of its stations and the time of the access method (T_{MAC}). Thus, the local BTF of an AP j is the sum of the busy time fractions due to transmissions to all of its stations.

$$b_j^L = \sum_{i \in S_j} b_{ij}^L$$

where S_j is the set of stations associated with AP j, and b_{ij}^L the BTF corresponding to the transmissions from AP j to station i. b_{ij}^L can be computed as follows:

$$b_{ij}^L = \overline{T(R_{ij}, L)} \times D_i$$

where $\overline{T(R_{ij}, L)}$ is the average time required for AP j to transmit one datagram of size L to station i with data rate R_{ij}. D_i is the average number of datagrams transmitted to station i in one second. It does not take into account retransmissions.

Nevertheless, a datagram is subject to transmission errors and may require one or more retransmissions. According to the IEEE 802.11 standard, the time required for AP j to successfully transmit a frame of size L to station i at data rate R_{ij} after k attempts is given by:

$$\begin{aligned} T(k, R_{ij}, L) &= T_{PHY} + T_{MAC} \\ T_{PHY} &= T_P + T_H + L/R_{ij} + T_{Ack} \\ T_{MAC} &= T_{DIFS} + T_{SIFS} + T_{backoff}(k) \end{aligned}$$

T_P and T_H represent the duration of the preamble and the header of the physical layer. T_{DIFS} is the DCF Inter Frame Space and T_{SIFS} is the Short Inter Frame Space. T_{ACK} is the duration of the ACK frame. $T_{backoff}(k)$ is the average back-off after k unsuccessful successive transmission attempts and is given by:

$$T_{backoff}(k) = \frac{min\left(2^k(CW_{min}+1)-1, CW_{max}\right)}{2} \times T_{slot}$$

where T_{slot} is the duration of a slot. CW_{min} and CW_{max} are respectively the minimum and maximum sizes of the contention window.

The average time that AP j requires to correctly transmit to station i or discard a single datagram is [22]:

$$\overline{T(R_{ij},L)} = p_{ij}T(0,R_{ij},L) + \sum_{k=1}^{m}\left(p_{ij}(1-p_{ij})^k\left(\sum_{l=0}^{k-1}T_c(l,R_{ij},L)+T(k,R_{ij},L)\right)\right) + (1-p_{ij})^{m+1}\sum_{l=0}^{m}T_c(l,R_{ij},L)$$

where m is the maximum number of retransmissions, p_{ij} is the probability of success to transmit from APj to station i and $T_c(l,R_{ij},L) = T_{backoff}(l) + T_{DIFS} + T_P + T_H + L/R_{ij} + T_{ATO}$ is the time between two consecutive transmissions if the frame transmission fails (T_{ATO} is the acknowledgment timeout).

Note that, in the previous formula, the value of the parameter R_{ij} may be different at each retransmission, but is constant during a retransmission. It is consistent with current implementations like Minstrel.

2) Neighbor Busy Time Fraction: In this section, we present the model to estimate the fraction of time the channel is sensed busy by an AP due to the activity of the other APs. Formally, in Wi-Fi networks, channel sensing is performed by the CCA (Clear Channel Assessment) mechanism [1]. We consider a conflict graph [23] composed of vertices that represent APs and where an edge (j,k) exists if APs j and k detect their mutual transmissions (according to the CCA mechanism).

To compute the Neighbor BTF b_j^N, we introduce a set of notations. We define the event A_i as follows:

$$A_i = \{\text{AP } i \text{ is transmitting}\} \tag{1}$$

So, the fraction of time that the channel is sensed busy by AP j due to transmissions from its neighbors is given by:

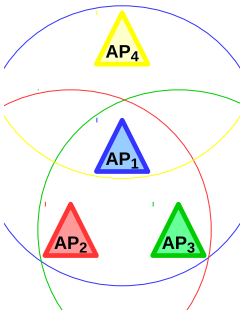
$$b_j^N = Pr\left(\bigcup_{i\in N_j} A_i\right) \tag{2}$$

where N_j is the set of neighbors of vertex j in the conflict graph. The events A_i are not disjoint and the computation of the union is consequently not trivial. To compute this probability, we propose to use the **Inclusion-Exclusion Principle** [24], which is defined as:

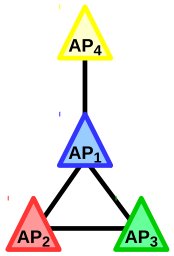
$$Pr\left(\bigcup_{i\in N_j} A_i\right) = \sum_{k=1}^{|N_j|}\left((-1)^{k-1}\sum_{\substack{I\subset N_j\\|I|=k}} Pr\left(\bigcap_{l\in I} A_l\right)\right) \tag{3}$$

where $I \subset N_j$ with $|I| = k$ describes all the subsets of N_j with cardinal k.

If $|I| = 1$, the computation $Pr\left(\bigcap_{l\in I} A_l\right)$ is trivial. When $|I| > 1$, we have to take into account the conflict graph. Indeed, there are two possible cases that we illustrate through the example given in Figure 1. We consider BTF of AP 1. It has three neighboring APs. As there is a link between AP 2 and AP 3, they cannot transmit at the same time and $Pr(A_2 \cap A_3) = 0$. As there is no conflict between AP 3 and AP 4, transmissions from these APs can overlap and $Pr(A_3 \cap A_4) \neq 0$.



(a) APs Topology



(b) APs conflict graph

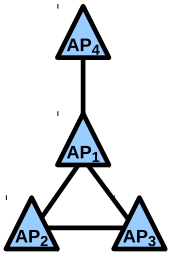
Fig. 1. Topology with 4 APs and its conflict graph

Consequently if two APs in I are neighbors, then their transmissions are exclusive and the intersection is zero:

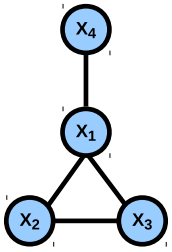
$$Pr\left(\bigcap_{l\in I} A_l\right)_{\substack{I\subset N_j\\|I|=k}} = 0, \text{ if } \exists (p,q)\in I^2 \text{ s.t. } p\in N_q (q\in N_p) \tag{4}$$

Otherwise, transmissions are not exclusive and this probability may be > 0.

But, the events $(A_l)_{l\in I}$ are not independent and the probability of their intersection (overlap) cannot be computed as their product. In order to compute this probability, we consider this problem as an **Undirected Graphical Model** or **Markov Network**. It is based on a graph where the vertices correspond to the events A_i and the edges represent the dependencies between them.



(a) APs conflict graph



(b) Markov Network

Fig. 2. 4 APs conflict graph and its Markov Network. Formally, the Markov Network is defined as function of the correlations between random variables. So, we introduce the random variables $(X_j)_j$ which indicates if AP j is transmitting ($A_j = \{X_j = 1\}$).

The considered graph is then the same as the APs conflict graph. In Figure 2 we show the previous example with a

topology with 4 APs conflict graph and the corresponding Markov Network.

Markov network relies on the Global Markov Property [25], which is defined as follows:

Definition: For any disjoint subsets of the vertices A, B, and C in the graph G such that C separates A and B (i.e. every path between a node in A and a node in B passes through a node in C), the random variables X_A are conditionally independent of X_B given X_C, i.e. $X_A \perp X_B / X_C$, where $X_A = \{X_v\}_{v \in A}$.

In our context, we assume that the transmissions of non-neighboring APs are independent if the set of all their neighbors (union of neighbors) does not transmit:

$$Pr\left(\bigcap_{l\in I} A_l\right) = Pr\left(\bigcap_{l\in I} A_l \middle| \bigcap_{l'\in I'} \overline{A_{l'}}\right) Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right), \quad I \subset N_j, |I| = k, I' = \bigcup_{i\in I} N_i$$

Applying this property, we obtain:

$$\begin{aligned} Pr\left(\bigcap_{l\in I} A_l\right) &= \prod_{l\in I} \left(Pr\left(A_l \middle| \bigcap_{l'\in I'} \overline{A_{l'}}\right)\right) Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right) \\ &= \left[\prod_{l\in I} \frac{Pr\left(A_l \cap \left(\bigcap_{l'\in I'} \overline{A_{l'}}\right)\right)}{Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right)}\right] Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right) \\ &= \frac{\prod_{l\in I} Pr\left(A_l \cap \left(\bigcap_{l'\in I'} \overline{A_{l'}}\right)\right)}{\left(Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right)\right)^{|I|-1}} \end{aligned} \quad (5)$$

Moreover, we have,

$$Pr\left(A_l \cap \left(\bigcap_{l'\in I'} \overline{A_{l'}}\right)\right) = Pr\left(A_l \cup \left(\bigcup_{l'\in I'} A_{l'}\right)\right) - Pr\left(\bigcup_{l'\in I'} A_{l'}\right) \quad (6)$$

and,

$$Pr\left(\bigcap_{l'\in I'} \overline{A_{l'}}\right) = 1 - Pr\left(\bigcup_{l'\in I'} A_{l'}\right) \quad (7)$$

By substituting (6) and (7) in Equation (5), we obtain:

$$Pr\left(\bigcap_{l\in I} A_l\right) = \frac{\prod_{l\in I}\left(Pr(\bigcup_{l'\in I'\cup\{l\}} A_{l'}) - Pr(\bigcup_{l'\in I'} A_{l'})\right)}{\left(1 - Pr(\bigcup_{l'\in I'} A_{l'})\right)^{|I|-1}} \quad (8)$$

We obtain a system of nonlinear equations where each term (variable) is the probability of union of the events $\{A_i\}$. As the number of possible combinations between all events is finite then the system contains a finite number of equations. This system can be solved by any numerical method.

To sum up, b_j^N, given by Equation (2), is obtained from the union of the events A_i (Equation (3)), itself obtained from Equation (8).

IV. Association optimization

Our association scheme is based on BTF. This quantity describes the saturation level of an AP. If an AP is saturated, its BTF is close to 1, and the associated stations are likely restrained in terms of throughput and thus unsatisfied. On the other hand, if BTF is lower, stations necessarily obtain the required throughput since a part of the bandwidth is available and not used. Stations are then assumed satisfied in terms of their throughput demand. The optimization problem aims to minimize the maximum BTF in the network. Formally, it is given by Equation (9).

Algorithm 1 BTF association algorithm

//Initialization
Collect measurements from APs
- station data rates R_{ij}
- station success probabilities p_{ij}
- APs BTF b_j

Infer the APs conflict graph for each channel
$Max_{BTF} = \max_{j\in A}[b_j]$
//The optimization loop
while Convergence() = false **do**
  for all Sta i **do**
    for all APs j **do**
      //check if Sta i lies in the transmission range of AP j
      if $R_{ij} \neq 0$ **then**
        Compute the success probability p_{ij} with AP j
        //We do not associate Sta i to AP j if it will be saturated
        if $b_j + b_{ij}^L < 1$ **then**
          associate Sta i with AP j
          compute new BTF for all APs: $b_j = b_j^L + b_j^N$
          if $\max_{i\in A}[b_i] < Max_{BTF}$ **then**
            save the best value: $Max_{BTF} = \max_{i\in A}[b_i]$
            save this best association change
          end if
          cancel this change
        end if
      end if
    end for
  end for
  Apply the best association change
end while
end procedure

$$minimize \ \max_{j \in A} [b_j] \quad (9)$$

where A is the set of APs and b_j the BTF of AP j. This objective function has been designed to:

- share the load among APs as it will try (if such solutions exist) to unload the most loaded APs,
- satisfy a maximum of stations in terms of throughput, as it will try to decrease BTF of saturated APs,
- increase the station throughputs, as unsatisfied stations will be moved to unsaturated APs.

The evaluation of BTF of each AP relies on the model proposed in Section III-B which predicts b_j (i.e. BTF of AP j) for any association. We propose an iterative heuristic based on the principle of local search to solve our optimization problem. Local search is an important class of heuristics used to solve combinatorial optimization problems. The key idea of a local search algorithm is to start from an initial feasible solution (association) and iteratively find, at each iteration, a solution called a best neighbor that improves the objective function [26]. The main benefits of local search lie in its simplicity and its iterative process which can stop the optimization process at any time to comply with a constraint like the computation time for instance. This is supported by the fact that the local search algorithms consider only complete feasible solutions during the search. The proposed algorithm has then the advantage to improve Wi-Fi associations at each iteration, and can be stopped at any time with a feasible solution. The time that the system spends in computing a solution can thus be bounded and tuned.

The controller runs the iterative local search Algorithm 1, which consists in:

1) starting with an initial configuration (RSSI or any current association),
2) then at each iteration, it chooses, among all possible association changes, the one that minimizes the objective function. This configuration becomes the current solution on which to apply the next iteration.

V. Evaluation

In this section, we present the simulation environment, the performance metrics, and the different simulation scenarios. We then discuss the simulation results.

A. Simulation configuration

We used a fixed point method [27] to solve the system of nonlinear equations. The optimization heuristic is implemented using C++ programming language. Simulations have been performed with the network simulator 3 (ns-3). We use the log-distance path loss model. The transmission power is set to 16 dbm. The number of APs and stations is fixed for each topology. Stations are associated according to the RSSI value in the initial configuration. The ideal Wi-Fi manager of ns-3 is used to determine the data rates between APs and stations. In the figures, we increase the input rates (mean of the flow rates between APs and stations). For each input rate, simulations are repeated 30 times with different stations position. Flow rates are constant during a simulation but set randomly with a given average. Therefore, physical transmission rates and flows are different from a station to another for each simulation. A 95% confidence interval is computed over these 30 samples. The duration of each simulation is 60 seconds.

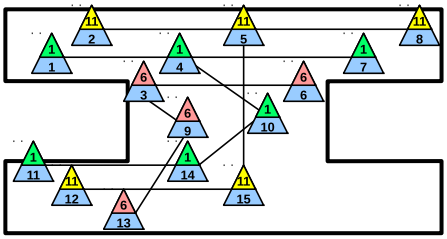


Fig. 3. WLAN topology at one floor in our university. The upper number corresponds to the used channel and the lower number corresponds to the AP number.

B. BTF Estimation

In order to estimate the accuracy of the BTF model proposed in Section III-B we compare its values obtained by simulation and from the model. The considered topology is the WLAN of our university at a given floor of the building. This network is composed of 15 APs as shown in Figure 3. APs use the ISM frequency band (2.4 Ghz). In this band the number of non-overlapping channels is limited (three orthogonal channels: 1, 6 and 11). This Figure shows also the three conflict graphs (one for each orthogonal channel).

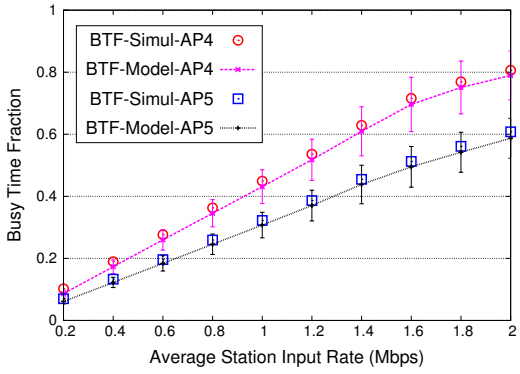
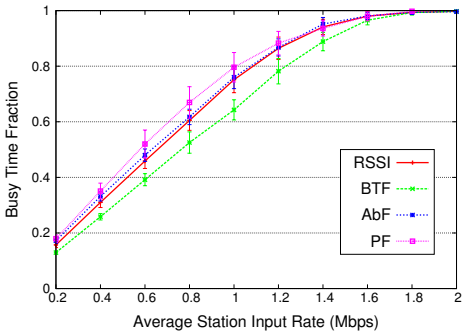


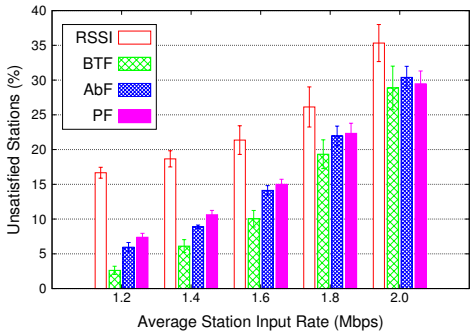
Fig. 4. Simulation vs Model BTF values for APs 4 and 5 in the university topology

The simulated scenario consists of 60 stations randomly distributed in the coverage area of the 15 APs. We plot in Figure 4 the BTF values according to the average input rate (mean of the flow rates). To evaluate the effectiveness of the approach in dense environment, we consider BTF of APs 4 and 5 of our topology (each one is in conflict with 3 other APs).

For AP 4, the difference between the measured BTF (simulation) and model is 7% when the load is low ($BTF < 0.2$). This difference decreases to 2% when the load increases



(a) BTF of the most loaded AP



(b) Unsatisfied stations

Fig. 5. BTF of the most loaded AP and proportion of unsatisfied stations according to the WLAN load.

($BTF > 0.7$). For AP 5 the difference varies in average between 3% and 6%. According to these results, it appears that the model provides a very good estimation of BTF. The model slightly underestimates the BTF as we do not take into account acknowledgments transmitted by the stations. It is neglected because to include them in the BTF computation a full knowledge of the conflict graph is required (in particular conflicts between stations) whereas in our model only conflicts between APs are considered. It is more realistic from an implementation point of view, but these conflicts/acknowledgments can be easily integrated in the model if the controller is able to infer them.

C. Association Optimization

In order to evaluate the improvement brought by our approach, we have considered two different topologies. From the simulation results, we compute the following performance parameters:

- Busy Time Fraction: for each simulation we consider the greatest BTF in the network.
- Number of unsatisfied stations: it represents the proportion of stations that are not satisfied in terms of throughput (i.e. when the ratio between the obtained throughput and the demand is less than 98%).
- Throughput Satisfaction Ratio: it is the ratio between the throughput obtained and the throughput requested for each station.

Our solution is compared to three existing approaches:

- Initial configuration: stations associate to APs according to the value of the RSSI. It is denoted as RSSI in the figures.
- Access based Fairness [5]: stations associated to the same AP have the same opportunity of service in saturated mode. It is denoted AbF in the figures.
- Proportional Fairness [7]: the saturated mode is also considered with an access opportunity to the medium which is proportional to the data rate of each station. It is denoted PF in the figures.

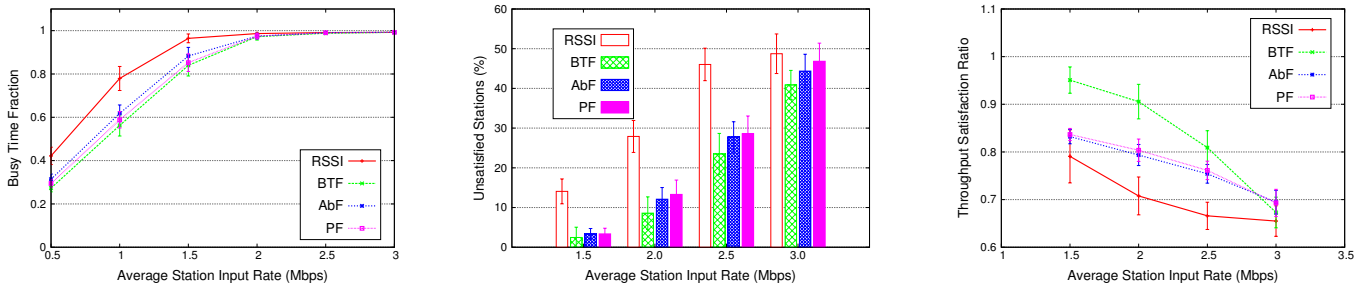
In the performance evaluation we consider different flow types, as follows:

- UDP: all the packets have the same size (1500 bytes) and the inter-packet time is constant for each station.
- Real trace: packet sizes vary according to a distribution obtained from a real trace [28] (Average Packet Size = 755.572 bytes and Standard Deviation = 674.05).
- TCP: constant bit rate flows are installed over TCP connections.

a) ENS topology.: The first scenario considers the topology of our university (ENS) used in the previous section and UDP flows. Figure 5a illustrates the BTF of the most loaded AP as function of the WLAN load. For AbF, the busy time fraction is approximately the same as the one observed for the RSSI association. For PF association, the busy time fraction is even increased of 11% in average. With BTF optimization, the busy time fraction is decreased by approximately 15% when the network is not heavily loaded. This will allow stations to request more traffic without saturating APs. It appears clearly that AbF and PF approaches are unable to decrease BTF in unsaturated WLAN.

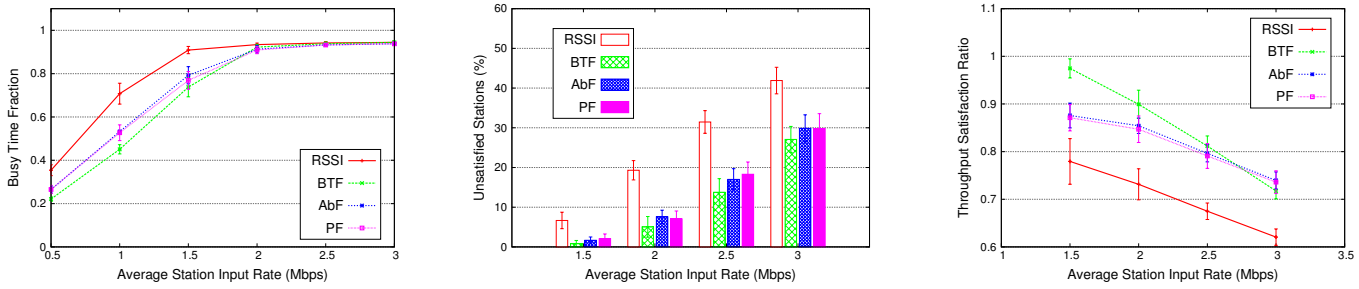
Nevertheless, when the WLAN reaches saturation, the three approaches provide similar results in terms of busy time fraction. To show the benefit of our approach when the WLAN becomes loaded (more than 1 Mbps per station in the figure), we measure and compare the number of stations not satisfied before and after the optimization for the three approaches. Results are shown in Figure 5b. Our solution reduces the number of unsatisfied stations by more than 84% for an average load of 1.2 Mbps per station, and 18% for an average load of 2 Mbps per station. For AbF, the gain varies between 64% and 14%, and for PF it varies between 55% and 16%. Even in saturated conditions, our solution still presents a lower number of unsatisfied stations.

b) Random topologies.: To evaluate our approach with denser topology and more complex conflict graphs between APs, we performed simulations on random topologies. Each topology is composed of 25 APs uniformly deployed in a square of size $500m \times 500m$. 100 stations are distributed in the coverage area of these APs. APs are configured with 8 orthogonal channels. In this scenario, APs location is changed at each simulation. This randomness allows us to consider an



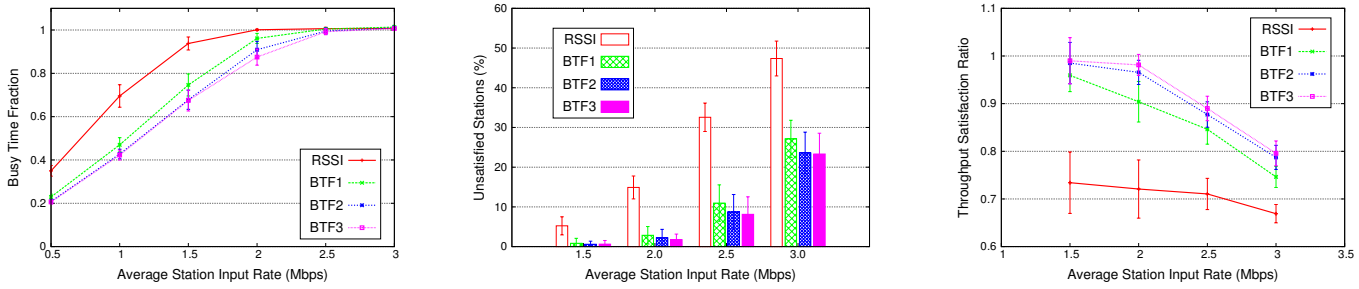
(a) BTF of the most loaded AP (b) Unsatisfied stations (c) Throughput Satisfaction Ratio

Fig. 6. BTF association optimization using random topology with the Real Trace flows



(a) BTF of the most loaded AP (b) Unsatisfied stations (c) Throughput Satisfaction Ratio

Fig. 7. BTF association optimization using random topology with TCP flows



(a) BTF of the most loaded AP (b) Unsatisfied stations (c) Throughput Satisfaction Ratio

Fig. 8. BTF association optimization using random topology with UDP flows and applied 3 times

important number of different topologies (30 for each set of parameters).

Figure 6 illustrates the results of the simulations with the real trace. It allows us to evaluate performances for different packet sizes. Figure 6a shows that the BTF approach can offload 35% of the most charged AP. AbF and PF approaches allow a load reduction of about 25% and 30% respectively. In Figure 6b, the number of unsatisfied stations is decreased of 54% with BTF optimization with regard to to the RSSI association. For AbF and PF, improvements are 45% and 42% in average. The satisfaction ratio is shown in Figure 6c. For the BTF approach, it is increased in average between 3% and 27%. On the other hand, AbF approach allows a gain between 6% and 13%, and between 5% and 14% for PF.

Figure 7 shows results with TCP flows. In Figure 7a our approach decreases the load of the most loaded AP up to 37%. For the AbF and PF approaches the decrease is almost the same and does not exceed 25%. Regarding the number of unsatisfied stations shown in Figure 7b, BTF allows a gain between 87% and 35%. For AbF, the decrease of the number of unsatisfied stations varies between 79% and 28%. For PF, the decrease is between 69% and 29%. Figure 7c plots the throughput satisfaction ratio. With BTF the stations gain in throughput on average between 25% and 15%. The AbF and PF approaches allow an average gain of 17% and 16% respectively.

In order to illustrate the impact of the BTF approach in a more realistic implementation context where the optimization process is executed whenever needed (at regular interval for instance), we have simulated the same scenario with UDP flows

in which our optimization method is applied 3 times.After each optimization we evaluate the performance and then collect the necessary measures for the next optimization. Results for this scenario are shown in Figure 8.

Even if the first optimization allows significant improvements for all performance parameters, the second and third optimizations can further improve these performances. For the greatest BTF in the network, the improvement for the first, second and third optimizations is in average 30%, 35% and 36% respectively. For the unsatisfied station number, the improvement is 68%, 74% and 75% respectively. For the throughput satisfaction ratio, the improvement is 22%, 27% and 29% respectively.

All these results tend to show that our solution generally offers better performance whatever the load of the network. Nevertheless, when the network is very loaded (average station input rate > 2.5 Mbps) the AbF and PF approaches allow to have results close to BTF.

VI. Conclusion

In this paper, we propose an original approach for the association optimization in Wi-Fi networks. Our solution is based on a model predicting BTF at each AP and aims to associate stations in order to minimize the most loaded AP. We have shown through simulations that the model allows an accurate estimation of BTF in the considered configurations. Moreover, performance evaluation has shown that such an approach reduces the congestion in the network as it decreases the most loaded AP. This improvement can reach 18% in average when the network is not heavily loaded. Also, our solution decreases the number of unsatisfied stations, up to 80% when the network becomes saturated and improves throughput of the unsatisfied stations. When the network is unsaturated, which corresponds to the normal conditions of a Wi-Fi network, approaches based on models that rely on saturated conditions are significantly less efficient that our proposition.

Acknowledgment

The authors wish to thank Isabel Martin Faus for her thorough and constructive comments of an earlier version of this paper.

References

[1] "IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Std 802.11-2012, pp. 1–2793, March 2012.

[2] P. Calhoun, M. Montemurro, and D. Stanley, "Control and provisioning of wireless access points (CAPWAP) protocol specification," Internet Requests for Comments, RFC Editor, RFC 5415, March 2009.

[3] K. Sood, S. Liu, S. Yu, and Y. Xiang, "Dynamic access point association using software defined networking," in 2015 International Telecommunication Networks and Applications Conference (ITNAC), Nov 2015, pp. 226–231.

[4] IEEE Std 802.11v-2011 Amendment 8: IEEE 802.11 Wireless Network Management, pp. 1–433, Feb 2011.

[5] M. Amer, A. Busson, and I. Guérin Lassous, "Association optimization in wi-fi networks: Use of an access-based fairness," in Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ser. MSWiM '16. ACM, 2016, pp. 119–126.

[6] H. Tang, L. Yang, J. Dong, Z. Ou, Y. Cui, and J. Wu, "Throughput optimization via association control in wireless LANs," Mobile Networks and Applications, vol. 21, no. 3, pp. 453–466, 2016.

[7] O. B. Karimi, J. Liu, and J. Rexford, "Optimal collaborative access point association in wireless networks," in IEEE INFOCOM 2014. Conference on Computer Communications, April 2014, pp. 1141–1149.

[8] IEEE 802.11k-2008 Amendment 1: IEEE 802.11 Radio Resource Measurement, 2008.

[9] I. Sohn, "Access point selection game with mobile users using correlated equilibrium," PLOS ONE, vol. 10, no. 3, pp. 1–13, 03 2015.

[10] Z. Chen, Q. Xiong, Y. Liu, and C. Huang, "A strategy for differentiated access service selection based on application in wlans," in 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), April 2014, pp. 317–322.

[11] J. B. Ernst, S. Kremer, and J. J. P. C. Rodrigues, "A utility based access point selection method for ieee 802.11 wireless networks with enhanced quality of experience," in 2014 IEEE International Conference on Communications (ICC), June 2014, pp. 2363–2368.

[12] M. V. Ramesh and M. S. Nisha, "Design of optimization algorithm for wlan ap selection during emergency situations," in 2011 IEEE 3rd International Conference on Communication Software and Networks, May 2011, pp. 340–344.

[13] W. Wong, A. Thakur, and S. H. G. Chan, "An approximation algorithm for ap association under user migration cost constraint," in IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, April 2016, pp. 1–9.

[14] E. Zola, F. Barcelo-Arroyo, and A. Kassler, "Multi-objective optimization of wlan associations with improved handover costs," IEEE Communications Letters, vol. 18, no. 11, pp. 2007–2010, Nov 2014.

[15] K. Khawam, J. Cohen, P. Muhlethaler, S. Lahoud, and S. Tohme, "Ap association in a ieee 802.11 wlan," in 2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Sept 2013, pp. 2142–2147.

[16] P. B. Oni and S. D. Blostein, "Ap association optimization and cca threshold adjustment in dense wlans," in 2015 IEEE Globecom Workshops (GC Wkshps), Dec 2015, pp. 1–6.

[17] A. Ozyagci, K. W. Sung, and J. Zander, "Association and deployment considerations in dense wireless lans," in 2014 IEEE 79th Vehicular Technology Conference (VTC Spring), May 2014, pp. 1–5.

[18] M. Abusubaih, S. Wiethoelter, J. Gross, and A. Wolisz, "A new access point selection policy for multi-rate ieee 802.11 wlans," Int. J. Parallel Emerg. Distrib. Syst., vol. 23, no. 4, pp. 291–307, Aug 2008.

[19] B. P. Tewari and S. C. Ghosh, "Interference avoidance through frequency assignment and association control in ieee 802.11 wlan," in 2014 IEEE 13th International Symposium on Network Computing and Applications, Aug 2014, pp. 91–95.

[20] D. Gong and Y. Yang, "On-line ap association algorithms for 802.11n wlans with heterogeneous clients," IEEE Transactions on Computers, vol. 63, no. 11, pp. 2772–2786, Nov 2014.

[21] F. Malandrino, C.-F. Chiasserini, and S. Kirkpatrick, "Cellular Network Traces Towards 5G: Usage, Analysis and Generation," IEEE Transactions on Mobile Computing, vol. PP, no. 99, p. 1, August 2017.

[22] V. Atanasovski and L. Gabrilovska, "Influence of header compression on link layer adaptation in ieee 802.11b," in 2005 International Conference on Wireless Networks, Communications and Mobile Computing, vol. 2, June 2005, pp. 1551–1556 vol.2.

[23] A. Busson, E. Fleury, and N. M. Phung, "A simple method to infer Wi-Fi conflict graph," in CORES2017, Quiberon, France, May 2017.

[24] T. Andreescu and Z. Feng, Inclusion-Exclusion Principle. Boston, MA: Birkhäuser Boston, 2004, pp. 117–141.

[25] D. Edwards, Introduction to Graphical Modelling, ser. Springer Texts in Statistics. Springer New York, 2012.

[26] J. P. Walser, Integer Optimization by Local Search: A Domain-independent Approach. Berlin, Heidelberg: Springer-Verlag, 1999.

[27] V. Berinde, Iterative Approximation of Fixed Points, ser. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 2007.

[28] "The CAIDA anonymized internet traces 2015 dataset," http://www.caida.org/data/passive/passive_2015_dataset.xml, 2015.

Carrier-Sense Multiple Access with Transmission Acquisition (CSMA/TA)

Marcelo M. Carvalho
Electrical Engineering Department
University of Brasília
Brasília, DF, Brazil 70919-970

J.J. Garcia-Luna-Aceves
Computer Engineering Department
UC Santa Cruz, Santa Cruz, CA, 95064
PARC, Palo Alto, CA 94304

***Abstract*—This paper introduces Carrier-Sense Multiple Access with Transmission Acquisition (CSMA/TA) for wireless local area networks (WLANs) with stations endowed with half-duplex radios using single antennas. In contrast to traditional contention-based channel-access methods, CSMA/TA seeks to increase the likelihood of having the last transmission from a group of colliding transmissions succeed. To accomplish this, a station senses the channel before sending a pilot packet. After finishing the transmission of the pilot, the station is required to wait for a certain amount of time before sensing the channel again. If the channel is sensed to be idle again, the station understands that "it has acquired its right to transmit a data frame" and proceeds with that. The throughput of CSMA/TA is compared with the throughputs of CSMA and CSMA/CD. An important feature of the analysis presented in this paper is the consideration of the impact of the receive-to-transmit and transmit-to-receive turnaround times. It is shown that CSMA/TA performs better than ideal CSMA and CSMA/CD if the propagation delays in the network are larger than the turnaround times, and its performance can still surpass CSMA/CD and CSMA if turnaround times are larger than propagation delays but not too much larger.**

I. Introduction

In the past few decades we have witnessed the explosive deployment of wireless networks worldwide, which has caused a dramatic change in the way people use the Internet and its many services by virtue of mobile devices. In particular, the unprecedented success of wireless local area networks (WLANs) has allowed fast and easy connectivity in a number of environments, and its on-going evolution is now moving towards new realms, such as the Internet of Things (IoT), with long-range connections ($\sim$1000 m) at sub-GHz frequencies, typified by the latest IEEE 802.11ah standard [1]. However, while many technological advances have been incorporated into WLANs over the years, the most significant ones have been done at the physical layer, such as the adoption of advanced modulation and coding schemes, multiple-input multiple-output (MIMO) technologies, and wider channel bandwidths. By contrast, the core of the medium access control (MAC) sub-layer of current WLANs still relies on variations of the traditional carrier-sense multiple access (CSMA) technique first introduced by Kleinrock and Tobagi [2], as it is the case in the DCF used by stations allocated to a restricted access window (RAW) in the IEEE 802.11ah.

One of the key features of CSMA and many of its variants, such as CSMA/CD [3], is that all stations involved in a transmission collision are forced to give up and retry at a later time. Such an approach renders transmission periods during which the channel is wasted with packet collisions without resulting in any successful transmission. In CSMA/CD such wasted periods are shortened due to its full-duplex operation, by which a station monitors the channel while transmitting a frame, followed by its quick abortion if a collision is detected. Nevertheless, to date, CSMA/CD stands as the "holy grail" of contention-based MAC protocols for wireless networks, whose performance a number of works have tried to achieve using different techniques, such as multiple transceivers [4], [5] or the newest *full-duplex* radios based on self-interference cancellation [6], [7] (see Section II for related work).

The contribution of this paper is introducing **CSMA/TA** (Carrier Sense Multiple Access with Transmission Acquisition), which is a variant of CSMA for WLANs based on off-the-shelf *half-duplex* radios, and is such that the last transmission from a group of overlapping transmissions is allowed to succeed. The approach adopted in CSMA/TA leverages the short transmit-to-receive (TX/RX) and receive-to-transmit (RX/TX) turnaround times of modern half-duplex radios, which are about 2μs [8] and are far shorter than the 192μs incurred by other radios [9]. This is significant, because such turnaround times are of the same order of magnitude or even smaller than the propagation delays in many WLAN scenarios, especially those seeking long-range coverage.

Section III describes CSMA/TA. In a nutshell, a node that needs to send a data packet and senses the channel idle, first transmits a pilot packet, stops for a short time period to listen for other pilots, and if the channel is sensed to be idle during that time period it determines that the channel is free and proceeds to transmit the data packet accordingly. We call this process *transmission acquisition*. Section IV presents the throughput analysis of CSMA/TA, which is dictated by the relation between the propagation delay and the radio's TX/RX and RX/TX turnaround times. If the turnaround times are smaller than the propagation delay, then CSMA/TA guarantees that the node that transmitted the last pilot in a group of concurrent pilots *succeeds in acquiring the channel*, while the others back off. But, if the turnaround times are bigger than the propagation delay, the transmission acquisition depends on the *likelihood* of transmission acquisition which, in turn, depends on the relative magnitude of both aforementioned parameters.

Section V compares the throughput attained with CSMA/TA against CSMA and CSMA/CD in different scenarios, considering the impact of the turnaround times of half-duplex radios. As the results show, if the turnaround times are

ISBN 978-3-903176-08-9

much greater than the propagation delay, CSMA/TA performs slightly better than CSMA; however, if they are very close to the propagation delay CSMA/TA becomes more efficient than CSMA/CD. Section VI presents our conclusions.

II. Related Work

A number of contention-based channel access protocols have been proposed since CSMA [2] and CSMA/CD [3] were first introduced [13]. In particular, because collision detection using single-antenna half-duplex radios is not doable, CSMA/CD performance became the benchmark in the design of MAC protocols for wireless networks. Still, few proposals have been reported on how to emulate CSMA/CD using half-duplex radios. Rom [12] proposed a MAC protocol that detects collisions by means of pauses. A station that senses the channel busy defers transmission as in CSMA; a transmitter that senses the channel idle starts transmitting but pauses during transmission and senses the channel. If the channel is sensed idle, the sender completes its transmission; otherwise, the sender continues to transmit for a minimum transmission duration to jam the channel. This approach cannot guarantee that data packets will not collide with other transmissions at the receiver if packets start at the same time or the transmit-to-receive turnaround times are not negligible.

FAMA-PJ [11] emulates CSMA/CD in the context of collision avoidance in WLAN's and prevents data packets from colliding with other transmissions. A transmitter sends an RTS if it detects no carrier in the channel, and listens for a period of time after its RTS to check for jamming signals sent by passive nodes that detected a collision. A passive listener that receives the signal from the one or multiple RTS's sent and is unable to decode an RTS successfully sends a jamming signal for a period of time that is long enough to ensure that active transmitters hear the jamming signals from passive listeners once they can start listening to the channel after sending their RTS's. A remaining limitation of FAMA-PJ is that too many passive nodes end up sending jamming signals.

Other works have tried to emulate CSMA/CD by using more than one transceiver/antenna. For instance, Peng et al. [5] proposed a MAC protocol that requires two separate transceivers to operate on two separate channels for control and data frame transmissions. Pulses over the control channel are used for collision detection, along with a CTS frame to avoid hidden terminals. Also requiring two separate transceivers, CSMA/CN [4] utilizes the standard CSMA to acquire the medium. The intended receiver uses PHY-layer information to detect packet collisions, and notifies the transmitter via a distinct signature sent over the same data channel. The signature is unique to every transmitter, and the transmitter employs a separate, listener antenna to perform signature correlation to identify the notification. If the notification is identified, the transmitter aborts its transmission.

With the advent of single-channel full-duplex (FD) wireless transceivers based on self-interference cancellation (SIC) [10], a number of MAC protocols have been proposed to achieve CSMA/CD-like operation. For instance, FD-WiFi CSMA/CD [6] uses FD to implement carrier sensing while transmitting data. But, due to residual self-interference, the sensing threshold needs to be properly designed to balance the errors due to miss detection and false alarms. FD-CSMA/CD [7] implements a CSMA/CA with collision detection in which the receiver acknowledges the reception of a packet immediately if its header is correct, and keeps sending the ACK as long as no collision is detected. At the same time, the transmitting node keeps sending its packet as long as it keeps receiving the ACK. Thus, if the receiver detects a collision, it stops the ACK, which causes the transmitter to stop its transmission immediately. CSMA/CAD [14] also uses SIC to guarantee collision avoidance under hidden-terminal scenarios, while it implements collision detection during the four-way handshake. It is shown to attain higher throughput than CSMA, DBTMA, and CSMA/CA.

Although the potential of FD radios in the design of future MAC protocols is undeniable, the availability of cheaper half-duplex radios with much faster turnaround times allows the development of simple approaches that can even surpass the performance of CSMA/CD in certain conditions, and CSMA/TA is one alternative.

III. CSMA/TA

A. Motivation and Design Objectives

The operation of CSMA/TA is motivated by the observation that, to date, contention-based medium access control protocols have been designed under the premise that either: (a) all colliding stations should give up on their transmission attempt, no matter the order (and when) each colliding station started its attempt; or (b) stations can attempt to resolve collisions in a sequence of collision rounds. For instance, in CSMA and CSMA/CD, the first station to access the channel is forced to give up due to other stations who, inadvertently, initiated their transmission attempt at a slightly later time, causing frame collisions. Therefore, in such protocols, and the many variants that followed them, all stations are treated equally and are forced to retry at a later time, which leads to a waste of channel usage and, potentially, more channel contention.

But, what if a "winner" station could be named among a group of colliding stations? How would that be possible using only half-duplex radios with a single antenna? With that goal in mind, we designed CSMA/TA to allow the *last transmitting* station in a group of colliding stations to proceed with its data frame transmission, i.e., to implement the idea of the "last standing station always wins."

To accomplish the above, a station running CSMA/TA that has a data frame ready for transmission must first perform carrier sensing to check if the channel is clear. If the channel is clear, the station transmits a *pilot* packet that is common to every station participating in the network. The duration γ of a pilot must be greater than twice the maximum propagation delay τ in the WLAN. Once the transmission of the pilot is over, the sending station must simply *wait* for a period of time *equal to the propagation delay* τ. After waiting for τ seconds, the station executes carrier sensing again. If the channel is sensed to be idle, the station claims to have "acquired its right for transmission," and it immediately proceeds with the transmission of its data frame. Otherwise, if the channel is sensed to be busy, the station must refrain from transmitting its data frame and, consequently, must reschedule its transmission

to a future time according to some contention resolution algorithm, such as a back-off algorithm.

To illustrate the basic design idea in CSMA/TA, consider the case of three stations A, B, and C that are exactly within τ seconds from each other, as depicted in Figure 1. Station A senses the channel and finds it to be idle at time instant t_0; therefore, it initiates the transmission of its pilot of duration γ seconds. However, before A's pilot signal reaches stations B and C, i.e., before τ seconds elapse, stations B and C sense the channel at time instants t_B and t_C, respectively, and perceive the channel to be idle as well. Consequently, both stations B and C start transmitting their own pilots at $t_B, t_C \in (t_0, t_0+\tau]$. Once all stations complete the transmission of their pilots, they must all *wait* for τ seconds before sensing the channel again.

In this scenario, both A and B will refrain from transmitting their data frames because they will sense the channel busy after the waiting period of τ seconds. In the case of A, it will detect the presence of the pilots from both B and C, while B will detect the presence of the pilot from C, as indicated in the figure. Therefore, only station C will sense an idle channel after the waiting period of τ seconds, because it is the *last station who transmitted a pilot*. Consequently, C claims that it has *acquired the right to transmit its data frame*, and proceeds to transmit without collisions.

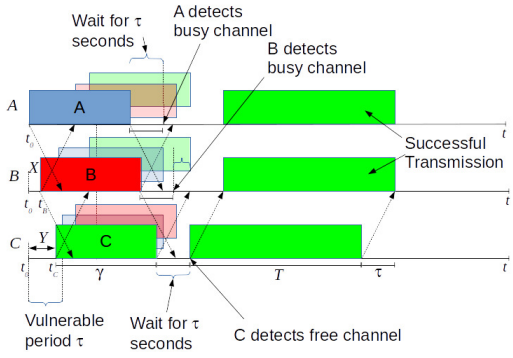


Fig. 1. CSMA/TA example with negligible turnaround latencies

More generally, if n stations initiate their pilot transmissions at *different* time instants in the interval $(t_0, t_0+\tau]$, where t_0 is the time instant where a reference station has first initiated its transmission, and assuming that $t_0 < t_1 < \ldots < t_{n-1} < t_n < t_0 + \tau$, where t_i is the time instant of the i-th pilot transmission then, after waiting for τ seconds after the end of their specific pilot transmission, only the n-th station *acquires the right for transmission*, while all other stations refrain from transmitting their data frames.

Unfortunately, the "wait for τ seconds before transmit" rule may not work if the transmit-to-receive (TX/RX) and the receive-to-transmit (RX/TX) turnaround times of the radios are taken into account. This is especially the case if such latencies are greater than the propagation delay in the WLAN; otherwise, the previous rule is valid. When that is the case, the *vulnerable period* for the occurrence of frame collisions increases, and we need to take that into account. The design of CSMA/TA considers these issues and their impact on the conditions for *transmission acquisition* to occur.

B. Non-negligible RX/TX and TX/RX Turnaround Times

To understand the impact of turnaround times on the operation of CSMA/TA and on the extension of the vulnerability periods surrounding any frame transmission, we go over another simple example. Let us assume that, at time instant t_0, a node A senses the channel, which means that its radio interface is in a state equivalent to a "receive" state. Let us also assume that node A perceives an idle channel at this same time instant, and immediately *starts the procedure* to initiate the transmission of its pilot. Before the pilot is actually transmitted, however, an *RX/TX turnaround time* of duration ε_1 seconds is incurred by the radio interface, followed by the pilot transmission itself, which lasts γ seconds. Once the pilot transmission is over, and following the CSMA/TA design, the station has to switch its radio interface to the *receive state* in order to sense the channel again. This incurs a *TX/RX turnaround time* that lasts ε_2 seconds, which is assumed to be greater than or equal to τ. Because of that, the rule of "wait for τ seconds before sensing the channel again" must be replaced by "wait for the TX/RX turnaround to finish." Then, all that is required is to *immediately sense the channel once the TX/RX turnaround time ε_2 is over*. Notice that, if $\varepsilon_2 < \tau$, we have a scenario that is equivalent to the rule of "wait for τ seconds before sensing the channel again."

Once station A switches to the receive mode, it senses the channel instantaneously. It is assumed that processing delays for carrier sensing or collision detection are negligible. If the channel is sensed to be idle again, the station may start the procedure to transmit a data frame, which will require an *additional RX/TX turnaround time* of duration ε_1, followed by the transmission of the data frame itself, which lasts T seconds. Finally, τ seconds are required for the complete data frame to reach every other node in the network. Figure 2 shows the time intervals incurred in the successful transmission of a data frame when no other station transmits.

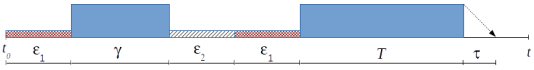


Fig. 2. Time diagram of a successful transmission of a data frame, including all time intervals involved in the process: RX/TX turnaround time ε_1, pilot duration γ, TX/RX turnaround time ε_2, RX/TX turnaround time ε_1, data frame T, and the propagation delay τ for the data frame to be received by all stations in the network completely.

In the previous scenario, a time interval of length $\varepsilon_1 + \tau$ seconds occurs from the instant when station A decides to transmit a pilot to the instant when that pilot first reaches the other nodes in the network (i.e., after a propagation delay). Hence, considering just another station B that has a data frame ready to be sent, and if it senses the channel at any time during the interval $(t_0, t_0 + \varepsilon_1 + \tau]$, station B will perceive an idle channel because A's pilot will not reach station B until the instant $t_0 + \varepsilon_1 + \tau$. Thus, the actual *vulnerability period*, i.e., the time interval during which stations can transmit without noticing other transmissions over the channel, *increases* from τ to $\tau + \varepsilon_1$ seconds. Then, depending on the time instant when station B starts transmitting its pilot, its signal may arrive at A while A is *still switching from transmit to receive mode*, as shown in Figure 3. If this happens, then when A finally switches to the receive mode, *it will perceive an idle channel*

similarly to B, in the end of its TX/RX turnaround time. In this case, both nodes will "claim their right to transmit" their data frames, and their frames will collide. Therefore, when the radio's turnaround times are taken into account, collisions may happen with the proposed CSMA/TA rule. Nonetheless, rather than insisting on the idea of having a successful station on *every group* of colliding stations, we will look at the conditions for the *likelihood of having a successful station* within a colliding group.

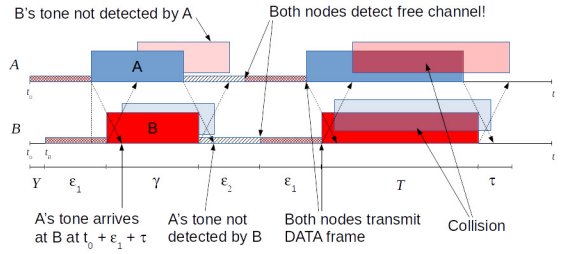


Fig. 3. Example showing transmission acquisition failing: Station A cannot perceive B's frame on the channel because, after finishing its own transmission, there is an extra time interval due to the TX/RX turnaround time before it can actually sense the channel. By the time the TX/RX turnaround time is over, the channel is clear. The same happens to B, and both A and B detect a free channel, which leads to the collision of data frames.

Let us explore the conditions for having station B successfully transmitting a data frame, i.e., to have station A refraining from transmitting its data frame, as in the "ideal" case. Let Y denote the length of the time interval between t_0 and the time instant when node B senses the channel and decides to transmit its pilot, i.e., its RX/TX turnaround time begins, as shown in Figure 4. In order for B to successfully acquire its right for transmission, node A must listen to the end of B's pilot (at least) when A's TX/RX turnaround time is over. This condition is depicted in Figure 4 when station A detects a busy channel.

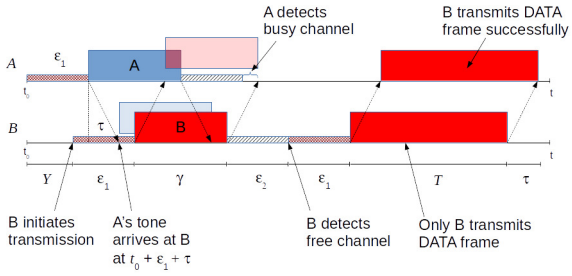


Fig. 4. Example of successful transmission acquisition. The time instant when B switches to transmit mode is such that the end of its pilot is sensed by the end of station A's TX/RX turnaround time.

Based on the above argument, the following inequality relating the relevant time intervals in the process must be always satisfied in order for A to refrain from transmitting its data frame and, consequently, have station B acquiring the channel for its data frame transmission without collision:

$$Y + \varepsilon_1 + \gamma + \tau > \varepsilon_1 + \gamma + \varepsilon_2 \;\Rightarrow\; Y > \varepsilon_2 - \tau, \qquad (1)$$

i.e., as long as Y is greater than $\varepsilon_2 - \tau$, station A is able to detect station B's pilot and refrains from transmitting its data frame. At the same time, if $Y > \varepsilon_1 + \tau$, station B detects A's pilot, and defers its transmission. Therefore, the length of the time interval Y that allows station B to acquire the channel successfully is bounded as follows:

$$\varepsilon_2 - \tau < Y \leq \varepsilon_1 + \tau. \qquad (2)$$

It follows from (2) that the RX/TX turnaround time ε_1 must be related to the TX/RX turnaround time ε_2 by

$$\varepsilon_1 > \varepsilon_2 - 2\tau, \text{ and } \varepsilon_1 \geq 0. \qquad (3)$$

If $\varepsilon_2 = \tau$, i.e., in the ideal case when there is no TX/RX turnaround time and the station has to wait for τ seconds before checking the channel again, the above inequality is satisfied with $\varepsilon_1 = 0$, i.e., when no RX/TX turnaround time is considered. This is exactly the scenario discussed in Section III-A.

Now, let us assume that $n \geq 1$ stations start their transmission procedures after a station A starts its transmission procedure at t_0, i.e., all stations start their transmission procedures in the interval $(t_0, t_0 + \varepsilon_1 + \tau]$, with the beginning of their RX/TX turnaround times at instants denoted by $t_1 < t_2 < \cdots < t_{n-1} < t_n$. Many scenarios are possible in this case. One such scenario is depicted in Figure 5, which shows three nodes A, B, and C starting their pilots at time instants t_0, t_B, and t_C, respectively. Assume that $X = t_B - t_0 > \varepsilon_2 - \tau$, and $Y = t_C - t_B < \varepsilon_2 - \tau$. So, in spite of having B initiating its transmission procedure at an instant that is distant apart from t_0 by more than $\varepsilon_2 - \tau$, station C starts its RX/TX turnaround time at an instant that does not follow the inequality *with respect to the last station that has initiated a transmission*, i.e, station B. As a result, stations B and C will perceive an *idle channel* at the end of their TX/RX turnaround time, and they will incur a collision of their data frames, as shown below.

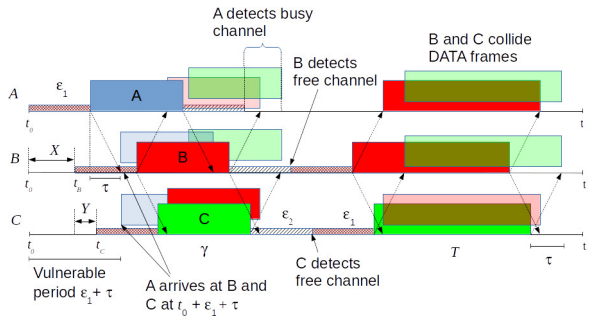


Fig. 5. CSMA/TA example with non-negligible turnaround times

Consider now the case when station C starts its transmission procedure at an instant t_C that is $\varepsilon_2 - \tau$ seconds apart from t_B, i.e., $t_C - t_B > \varepsilon_2 - \tau$. In this case, stations A and B will detect a busy channel, for sure, in the end of their TX/RX turnaround times, and they will defer their data frame transmissions. In this case, station C will *acquire the right for transmission*, and will transmit a data frame without collision, as it is shown in Figure 6.

It is important to notice that it is not enough to have *any* transmission initiation procedures apart from each other by $\varepsilon_2 - \tau$, but only the last and the next-to-the-last initiation procedures. Figure 5 clearly showed this situation, where A

and B are distant apart from each other by more than $\varepsilon_2 - \tau$ seconds, but B and C are not. In that case, B and C detected a free channel and collided.

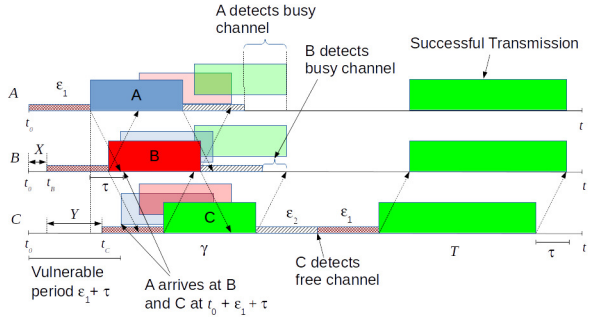


Fig. 6. Example of successful transmission acquisition when three stations compete for the channel. Station C is the last station to start its transmission procedures, and the time instant t_C is distant from the next-to-the-last station B by more than $\varepsilon_2 - \tau$ seconds. As a consequence, it acquires the channel.

Figure 7 shows a time diagram where the arrows indicate the time instants of transmission procedures of n stations within the time interval $(t_0, t_0 + \varepsilon_1 + \tau]$. In this case, the last time instant t_n must be such that $t_n - t_{n-1} > \varepsilon_2 - \tau$. Under such conditions, station n successfully acquires the channel.

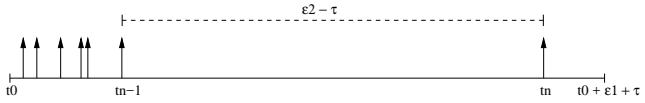


Fig. 7. Time instants of transmission procedures of n stations

IV. Throughput Analysis

We derive the normalized throughput of CSMA/TA for fully-connected networks under ideal channel conditions, and consider the impact of the RX/TX and TX/RX turnaround times. The performance of CSMA/TA is compared with non-persistent CSMA (with and without turnaround times), and CSMA/CD, which does not have turnaround times given that it requires full-duplex radios. We focus on the non-persistent versions of CSMA/TA, CSMA, and CSMA/CD using the traffic model first introduced by Kleinrock and Tobagi [2]. In this analysis, we do not consider the use of priority acknowlegdments (ACKs), because we are only concerned with errors due to multiple access interference.

According to our model, there is a large number of stations that constitute a Poisson source sending data packets to the the channel with an aggregate mean generation rate of g packets per unit time (i.e., new and retransmitted packets). Each node is assumed to have at most one data packet to be sent at any time, which results from the MAC layer having to submit one packet for transmission before accepting the next packet. A node retransmits after a random retransmission delay that, on the average, is much larger than the time needed for a successful transaction between a transmitter and a receiver and such that all transmissions can be assumed to be independent of one another. The channel is assumed to introduce no errors, so multiple access interference (MAI) is the only source of errors. Nodes are assumed to detect carrier perfectly.

We assume that two or more transmissions that overlap in time in the channel must all be retransmitted (i.e., there is no power capture by any transmission), and that any packet propagates to all nodes in exactly τ seconds. The RX/TX and TX/RX turnaround times at each radio interface are ε_1 and ε_2, respectively, and are assumed to be larger than or equal to the propagation delay τ, which agrees with the parameters assumed in IEEE 802.11 DCF. The transmission time of a data packet is T. For the case of CSMA/CD, the time of a jamming bit sequence is J, which is larger than the error-checking field of a packet (e.g., 48 bits). We assume that processing delays are negligible, which includes the time to detect carrier or do collision detection. The protocols are assumed to operate in steady state, with no possibility of collapse, and hence the average channel utilization of the channel is given by [2]

$$S = \frac{\overline{U}}{\overline{B} + \overline{I}}, \tag{4}$$

where $\overline{B}$ is the expected duration of a busy period, defined to be a period of time during which the channel is being utilized; $\overline{I}$ is the expected duration of an idle period, defined as the time interval between two consecutive busy periods; and $\overline{U}$ is the time during a busy period that the channel is used for transmitting user data successfully.

A. Non-Persistent CSMA/TA

Theorem: The throughput of non-persistent CSMA/TA is

$$S = \frac{Te^{-g(\varepsilon_2 - \tau)}}{\frac{1}{g} + 3\varepsilon_1 + 2\tau + \gamma + \varepsilon_2 + T - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1 + \tau)}\right]^2 + K}, \tag{5}$$

where $K = -(\varepsilon_1 + \tau)e^{-g(\varepsilon_1 + \varepsilon_2)}$.

Proof: Based on the discussion in Section III-B, event E, which denotes successful transmission acquisition, can be described by the union of two mutually exclusive events as follows:

$$E = \{\{\text{no transmissions} \in [t_0, t_0 + \varepsilon_1 + \tau]\} \cup \{\{\text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau]\} \cap \{t_n - t_{n-1} > \varepsilon_2 - \tau\}\}, \tag{6}$$

where the event $t_n - t_{n-1} > \varepsilon_2 - \tau$ requires that the interval between the last and next-to-the-last transmission attempt must be greater than $\varepsilon_2 - \tau$. Hence, the probability P_{suc} of *successful transmission acquisition* is given by

$$P_{suc} = P\{E\} = P\{\text{no transmissions} \in [t_0, t_0 + \varepsilon_1 + \tau]\} + P\{\{\text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau]\} \cap \{t_n - t_{n-1} > \varepsilon_2 - \tau\}\}, \tag{7}$$

where, due to our Poisson assumptions,

$$P\{\text{no transmissions} \in [t_0, t_0 + \varepsilon_1 + \tau]\} = e^{-g(\varepsilon_1 + \tau)}. \tag{8}$$

The second probability in (7) can be written as

$$P\{\text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau] \cap t_n - t_{n-1} > \varepsilon_2 - \tau\} = = P\{t_n - t_{n-1} > \varepsilon_2 - \tau \mid \text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau]\} \times \times P\{\text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau]\}, \tag{9}$$

where

$$P\{\text{some trans.} \in [t_0, t_0 + \varepsilon_1 + \tau]\} = 1 - e^{-g(\varepsilon_1 + \tau)}. \tag{10}$$

To simplify notation, let A = {some trans. $\in [t_0, t_0+\varepsilon_1+\tau]$}. Then, for the conditional probability in (9) we use total probability to get

$$\begin{aligned}
&P\{t_n - t_{n-1} > \varepsilon_2 - \tau \mid A\} = \\
&= \sum_{n=1}^{\infty} P\{t_n - t_{n-1} > \varepsilon_2 - \tau, N = n|A\} \\
&= \sum_{n=1}^{\infty} P\{t_n - t_{n-1} > \varepsilon_2 - \tau | N = n, A\} P\{N = n|A\}, \quad (11)
\end{aligned}$$

where N is the number of transmission attempts initiated in $(t_0, t_0+\varepsilon_1+\tau]$. Using Bayes' rule,

$$\begin{aligned}
P\{N = n|A\} &= \frac{P\{N=n, A\}}{P\{A\}} = \frac{P\{A|N=n\}P\{N=n\}}{P\{A\}} \\
&= \frac{P\{A|N=n\}[g(\varepsilon_1+\tau)]^n e^{-g(\varepsilon_1+\tau)}}{[1-e^{-g(\varepsilon_1+\tau)}]n!}, \quad (12)
\end{aligned}$$

which leads to

$$P\{N=n|A\} = \begin{cases} 0, & \text{if } N=0 \\ \frac{[g(\varepsilon_1+\tau)]^n e^{-g(\varepsilon_1+\tau)}}{[1-e^{-g(\varepsilon_1+\tau)}]n!}, & \text{if } N>0, \end{cases} \quad (13)$$

because $P\{A|N=n\} = 1$ if $N > 0$. From (11), we need to compute $P\{t_n - t_{n-1} > \varepsilon_2 - \tau | N = n, A\}$. For a Poisson process, the conditional probability density function of the first n count times, $T_1, T_2, \ldots, T_n$, given $\{N_{\Delta T} = n\}$, i.e., given that $N = n$ time instants have occurred in a given time interval ΔT, is given by [15]

$$f(t_1, t_2, \ldots, t_n | N = n) = \frac{n!}{(\Delta T)^n}, \quad (14)$$

if $0 < t_1 < \cdots < t_n < \Delta T$, and 0 elsewhere, where ΔT is the length of the time interval of interest, i.e., in our case, $\Delta T = \varepsilon_1 + \tau$. Using this fact, and since $0 < t_1 < t_2 < \cdots < t_{n-1} < t_n$, it can be shown that

$$\begin{aligned}
&P\{t_n - t_{n-1} > \varepsilon_2 - \tau | N = n, A\} = \\
&= \int_{\varepsilon_2-\tau}^{\varepsilon_1-\tau} \int_0^{t_n-\varepsilon_2+\tau} \int_0^{t_{n-1}} \cdots \int_0^{t_2} \frac{n!}{(\Delta T)^n} dt_1 \ldots dt_{t_{n-1}} dt_{t_n} \\
&= \left[\frac{\varepsilon_1 - \varepsilon_2 + 2\tau}{\varepsilon_1 + \tau}\right]^n. \quad (15)
\end{aligned}$$

Substituting (15), (13), (11), (10), and (8) into (7), we have

$$\begin{aligned}
P_{suc} &= e^{-g(\varepsilon_1+\tau)} + [1 - e^{-g(\varepsilon_1+\tau)}] \sum_{n=1}^{\infty} \frac{[(\varepsilon_1-\varepsilon_2)+2\tau]^n}{(\varepsilon_1+\tau)^n} \times \\
&\times \frac{[g^n(\varepsilon_1+\tau)^n] e^{-g(\varepsilon_1+\tau)}}{[1-e^{-g(\varepsilon_1+\tau)}]n!} \\
&= e^{-g(\varepsilon_1+\tau)} + e^{-g(\varepsilon_2-\tau)} \sum_{n=1}^{\infty} \frac{[g(\varepsilon_1-\varepsilon_2+2\tau)]^n}{n!} e^{-g(\varepsilon_1-\varepsilon_2+2\tau)} \\
&= e^{-g(\varepsilon_1+\tau)} + e^{-g(\varepsilon_2-\tau)} [1 - P\{N=0 \text{ in } (\varepsilon_1-\varepsilon_2+2\tau)\}] \\
&= \underbrace{e^{-g(\varepsilon_1+\tau)}}_{\text{no transmission in } [t_0, t_0+\varepsilon_1+\tau]} + \underbrace{e^{-g(\varepsilon_2-\tau)}}_{\text{no transmission within } (\varepsilon_2-\tau) \text{ s}} \\
&\times \underbrace{[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}]}_{\text{some transmission in the interval of length } (\varepsilon_1+\tau)-(\varepsilon_2-\tau)} \quad (16)
\end{aligned}$$

Finally,

$$\begin{aligned}
P_{suc} &= e^{-g(\varepsilon_1+\tau)} + e^{-g(\varepsilon_2-\tau)} \left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right] \\
&= e^{-g(\varepsilon_2-\tau)}, \quad (17)
\end{aligned}$$

which reduces to the fact that a successful transmission acquisition happens if the *last station* to transmit in $(t_0, t_0+\varepsilon_1+\tau]$ starts its transmission procedures within an interval that is at least $\varepsilon_2 - \tau$ seconds away from the next-to-the-last station in the same interval. Note that, if $\varepsilon_2 = \tau$, then $P_{suc} = 1$, regardless of the value of the propagation delay τ and the RX/TX turnaround time ε_1. Later, we will show that CSMA/TA has an *effective vulnerable period* that is $\varepsilon_1 - \varepsilon_2 + 2\tau$ seconds *smaller than Nonpersistent CSMA*, if we take into account the RX/TX turnaround time in CSMA as well. Given the P_{suc}, we can now proceed with the evaluation of $\overline{U}$, $\overline{B}$, and $\overline{I}$.

1) Average Busy Period: For the average busy period $\overline{B} = E[B]$, two events can happen: either a successful data frame transmission happens or a collision occurs. Therefore,

$$E[B] = E[B|\text{success}]P\{\text{success}\} + E[B|\text{fail}]P\{\text{fail}\}. \quad (18)$$

In the case of success, we need to consider the cases where either no one transmits in $[t_0, t_0+\varepsilon_1+\tau]$ (i.e., $N=0$), or one or more stations transmit in $[t_0, t_0+\varepsilon_1+\tau]$ (i.e., $N>0$), which leads to

$$\begin{aligned}
E[B|\text{success}] &= E[B|\text{success}, N=0]P\{N=0\} + \\
&+ E[B|\text{success}, N>0]P\{N>0\}. \quad (19)
\end{aligned}$$

For the first conditional probability, we have

$$\begin{aligned}
E[B|\text{success}, N=0] &= \varepsilon_1 + \gamma + \varepsilon_2 + \varepsilon_1 + T + \tau \\
&= 2\varepsilon_1 + \gamma + \varepsilon_2 + T + \tau, \quad (20)
\end{aligned}$$

while for the case $N > 0$ we have

$$\begin{aligned}
&E[B|\text{success}, N>0] = \\
&= E[Y + \varepsilon_1 + \gamma + \varepsilon_2 + \varepsilon_1 + T + \tau | \text{success}, N>0] \\
&= E[Y|\text{success}, N>0] + 2\varepsilon_1 + \gamma + \varepsilon_2 + T + \tau. \quad (21)
\end{aligned}$$

To compute $E[Y|\text{success}, N>0]$ we need to first notice that, given there is a success, the last node to transmit in the interval $[t_0, t_0+\varepsilon_1+\tau]$ must have actually transmitted within the interval $[t_0+\varepsilon_2-\tau, t_0+\varepsilon_1]$, because its transmission procedures must start at least $\varepsilon_2 - \tau$ seconds away from the next-to-the-last node in the interval. Therefore, for the last transmitting node, $\varepsilon_2 - \tau \le Y \le \varepsilon_1 + \tau$.

Let $Z = Y - (\varepsilon_2 - \tau)$. Then, $0 \le Z \le \varepsilon_1 - \varepsilon_2 + 2\tau$, and we can compute $F_Z(z) = P[Z \le z]$ by making

$$\begin{aligned}
P[Z \le z] &= P\{\text{no transmission in } [\varepsilon_1 - \varepsilon_2 + 2\tau - z]\} \\
&= e^{-g(\varepsilon_1-\varepsilon_2+2\tau-z)}. \quad (22)
\end{aligned}$$

Given that $Z \ge 0$, we can compute $E[Z]$ as follows

$$\begin{aligned}
E[Z] &= \int_0^{\infty} [1 - F_Z(z)] dz \\
&= \int_0^{\varepsilon_1-\varepsilon_+2\tau} 1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau-z)} dz \\
&= \varepsilon_1 - \varepsilon_2 + 2\tau - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right]. \quad (23)
\end{aligned}$$

Finally, because $Y = Z + (\varepsilon_2 - \tau)$,

$$E[Y|\text{success}, N>0] = E[Z] + \varepsilon_2 - \tau$$
$$= \varepsilon_1 + \tau - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right]. \quad (24)$$

Given $E[Y|\text{success}, N>0]$ we can substitute its value in (21) to compute $E[B|\text{success}, N>0]$, i.e.,

$$E[B|\text{success}, N>0] =$$
$$= 3\varepsilon_1 + 2\tau + \gamma + \varepsilon_2 + T - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right]. \quad (25)$$

Hence,

$$E[B|\text{success}] = (2\varepsilon_1 + \gamma + \varepsilon_2 + T + \tau)e^{-g(\varepsilon_1+\tau)} +$$
$$+ \left[1 - e^{-g(\varepsilon_1+\tau)}\right]\{3\varepsilon_1 + 2\tau + \gamma + \varepsilon_2 + T -$$
$$- \frac{1}{g}\left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right]\Big\}$$
$$= \underbrace{(2\varepsilon_1 + \gamma + \varepsilon_2 + T + \tau)}_{\text{length with no transmission}} +$$
$$+ \underbrace{\left\{\varepsilon_1 + \tau - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1-\varepsilon_2+2\tau)}\right]\right\}}_{\text{additional length due to some transmission in } [t_0, t_0+\varepsilon_1+\tau]} \times$$
$$\times \left[1 - e^{-g(\varepsilon_1+\tau)}\right] \quad (26)$$

To compute $E[B|\text{fail}]$ we notice that, in this case, the last transmission can happen anywhere in $(t_0, t_0 + \varepsilon_1 + \tau]$, which leads to

$$E[B|\text{fail}] = E[Y + \varepsilon_1 + \gamma + \varepsilon_2 + \varepsilon_1 + T + \tau|\text{fail}]$$
$$= E[Y|\text{fail}] + 2\varepsilon_1 + \gamma + \varepsilon_2 + T + \tau. \quad (27)$$

The computation of $E[Y|\text{fail}]$ can be obtained by first noticing that, in this case, $0 < Y < \varepsilon_1 + \tau$. Therefore, because arrivals are Poisson distributed,

$$F_Y(y) = P\{Y \le y\} = P\{\text{no transmission in } \varepsilon_1 + \tau - y\}$$
$$= e^{-g(\varepsilon_1+\tau-y)}. \quad (28)$$

Since Y is a non-negative random variable, we have

$$E[Y] = \int_0^\infty [1 - F_Y(y)]dy = \int_0^{\varepsilon_1+\tau} 1 - e^{-g(\varepsilon_1+\tau-y)}dy$$
$$= \varepsilon_1 + \tau - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1+\tau)}\right] \quad (29)$$

Therefore,

$$E[B|\text{fail}] = 3\varepsilon_1 + 2\tau + \gamma + \varepsilon_2 + T - \frac{1}{g}\left[1 - e^{-g(\varepsilon_1+\tau)}\right]. \quad (30)$$

Finally, the average busy time $E[B]$ is given by

$$\overline{B} = 3\varepsilon_1 + 2\tau + \gamma + \varepsilon_2 + T - (\varepsilon_1 + \tau)e^{-g(\varepsilon_1+\varepsilon_2)} -$$
$$- \frac{1}{g}\left[1 - e^{-g(\varepsilon_1+\tau)}\right]^2. \quad (31)$$

2) Average Idle Period ($\overline{I}$): The average length of an idle period I is simply the average inter-arrival time of packets, which are preceded by pilot transmissions, and this equals $1/g$ because inter-arrival times are exponentially distributed with parameter g.

3) Average Successful Busy Period: The average time period used to transmit useful data $\overline{U}$ is simply the useful portion of a successful busy period, which occurs with probability $P_{suc} = e^{-g(\varepsilon_2-\tau)}$.

Substituting the values of $\overline{U}$, $\overline{B}$, and $\overline{I}$ into (4) we obtain (5). □

Usually, it is more convenient to work with normalized values in the computation of the average throughput. Hence, if we normalize all time intervals with respect to the data frame transmission time T, i.e., if we make $a = \tau/T$, $b = \varepsilon_1/T$, $c = \varepsilon_2/T$, $d = \gamma/T$, and $G = gT$, then (5) becomes

$$S = \frac{Ge^{-G(c-a)}}{1 + (1 + 2a + 3b + c + d)G - \left[1 - e^{-(a+b)G}\right]^2 + K}, \quad (32)$$

where $K = -(a+b)Ge^{-(b+c)G}$. One special case of interest is the "ideal case," i.e., when $\varepsilon_1 = 0$ and $\varepsilon_2 = \tau$, i.e., $b = 0$ and $c = a$, which refers to the case when there are no turnaround times, and the rule "wait for τ" is employed. In this case,

$$S = \frac{G}{1 + (1 + 3a + d)G - [1 - e^{-aG}]^2 - aGe^{-aG}}. \quad (33)$$

B. Non-Persistent CSMA

The throughput for non-persistent CSMA is well-known [2]. If the RX/TX turnaround time is considered, however, the vulnerable period of CSMA increases to $\varepsilon_1 + \tau$. Therefore, if the ACKs are assumed to be received instantaneously through an ideal secondary channel, the normalized throughput becomes

$$S = \frac{Ge^{-(a+b)G}}{1 + [2(a+b)+1]G - [1 - e^{-(a+b)G}]^2 - K}, \quad (34)$$

where $K = (a+b)Ge^{-(a+b)G}$ and $b = \varepsilon_1/T$. Note that, if we consider the RX/TX turnaround time, the successful probability of CSMA considers an interval $\varepsilon_1 + \tau$ (or $a+b$) that is 2τ seconds bigger than the interval of CSMA/TA, which is $\varepsilon_2 - \tau$ (or $b - a$) in the case when $\varepsilon_1 = \varepsilon_2$. In other words, the successful probability of CSMA decays faster than CSMA/TA for non-negligible turnaround times.

C. Non-Persistent CSMA/CD

The throughput of non-persistent CSMA/CD under the previous assumption of instantaneous ACKs can be easily derived (see [16] without considering priority ACKs). There are no turnaround times in CSMA/CD because the stations can sense the channel while transmitting. Hence, if J denotes the jamming signal time duration, and $h = J/T$, then the normalized throughput is given by

$$S = \frac{Ge^{-aG}}{2 + (2a + h)G + Ge^{-aG}(1 - a - h - 1/G)}. \quad (35)$$

V. Numerical Results

We compare the performance of CSMA/TA with CSMA and CSMA/CD by considering different scenarios in terms of the data rate R, the transmission range r, and the packet size L. We assume that the TX/RX and RX/TX turnaround times are equal ($\varepsilon_1 = \varepsilon_2 = \varepsilon$) and fixed at 2μs. The CSMA/TA pilot signal is set to three times the propagation delay τ in every case, while the jamming signal of CSMA/CD has the same time duration J as its counterpart in Ethernet, i.e., 48-bit time, which favors CSMA/CD when propagation delays are longer.

The scenarios depict cases when the propagation delay is smaller than the turnaround times. Therefore, the modifier "ideal" in the graphs correspond to turnaround times that are smaller than the propagation delay, which we take into account by assuming a turnaround time of 0 for ideal CSMA/TA and CSMA. Such results (shown in dashed lines) are included to understand the impact of turnaround times on CSMA and CSMA/TA. We remind the reader that, if the propagation delay is *greater* than the turnaround times, CSMA/TA operates according to the *ideal* case, while CSMA still suffers the impact of the turnaround times. Hence, in long-haul coverage scenarios with propagation delays larger than or equal to 2μs, CSMA/TA would perform just as the "ideal CSMA/TA."

Figure 8 shows the results when $L = 1500$ bytes, $R = 1$ Mb/s and $r = 100$ m. In this case, the turnaround time $\varepsilon = 6\tau$. It is clear that *ideal* CSMA/TA achieves the best throughput, which increases monotonically to a value very close to 1.0 as the offered load G increases without bound (i.e., by taking the limit $G \to \infty$ in (33)). This behavior is in stark contrast to CSMA/CD and ideal CSMA, whose throughput values collapse as G increases, due to higher chances of frame collisions. If we consider the turnaround times, CSMA/TA performs slightly better than CSMA (solid lines), while CSMA/CD surpasses both of them. It is interesting to observe that, with turnaround times, the range of G values over which CSMA has non-zero throughput is almost an order of magnitude smaller than in the ideal case.

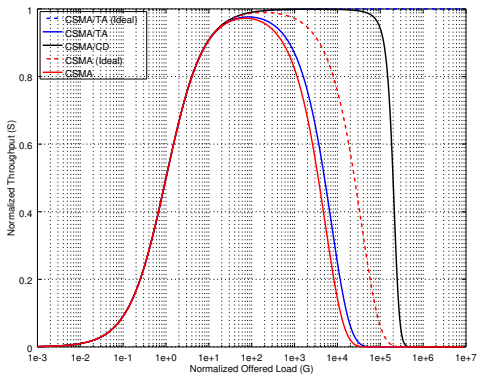


Fig. 8. S vs. G for $L = 1500$ bytes, $R = 1$ Mb/s, and $r = 100$ m.

Figure 9 shows the results when $L = 1500$ bytes, $R = 1$ Mb/s, and a turnaround time that is just 1% above the propagation delay, i.e., $\varepsilon = 1.01\tau$, which gives us $r = 594.06$ m. We can observe the great advantage of non-ideal CSMA/TA, whose throughput values not only match, but also surpass CSMA/CD at high loads. The results indicate that CSMA/TA can, in practice, accommodate a large number of devices that collectively generate a high traffic load (e.g., IoT scenarios). In this scenario, the likelihood of having a transmission acquisition within a group of colliding stations is high, as opposed to CSMA and CSMA/CD, who always force the whole group of colliding stations to retransmit in a future time.

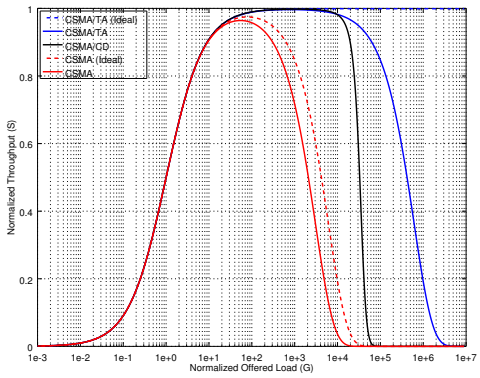


Fig. 9. S vs. G for $L = 1500$ bytes, $R = 1$ Mb/s, and $r = 594.06$ m.

Figure 10 shows the results for a data rate of 300 Mb/s with $L = 1500$ bytes, and an 100-m range. The performance of any protocol based on carrier sensing degrades as the ratio $a = \tau/T$ increases. Hence, the impact of the turnaround time is significant on both CSMA and CSMA/TA, and they achieve a maximum normalized throughput of about 0.6 and allow a much smaller range of traffic-load values, compared to the results of Figure 8. Here, the range of viable traffic-load values decreases by more than two orders of magnitude. We also notice that CSMA/TA performs slightly better than CSMA at high loads.

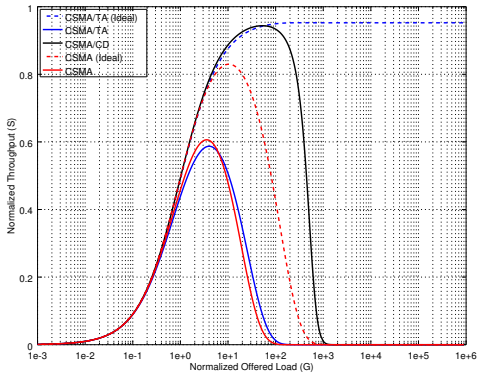


Fig. 10. S vs. G for $L = 1500$ bytes, $R = 300$ Mb/s, and $r = 100$ m.

Figure 11 shows the results for a data rate of 300 Mb/s, $L = 1500$ bytes, and a transmission range $r = 594.06$ m, i.e., $\varepsilon = 1.01\tau$. In this case, the maximum throughput of CSMA/TA is 0.68, which is 32% higher than the maximum throughput of CSMA, but just 8% smaller than CSMA/CD. At higher data rates, the overhead due to the pilot signal of CSMA/TA becomes more significant. In spite of that, CSMA/TA maintains throughput values above 0.6 for a wider range of traffic loads compared to CSMA/CD.

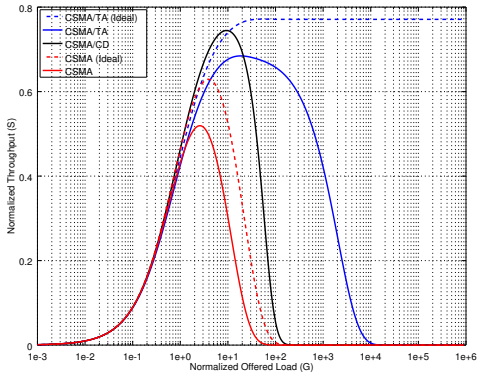


Fig. 11. S vs. G for $L = 1500$ bytes, $R = 300$ Mb/s, and $r = 594.06$ m.

Figure 12 shows the results for $r = 100$ m ($\varepsilon = 6\tau$) and Figure 13 shows the results for $r = 594.06$ m ($\varepsilon = 1.01\tau$) when $L = 100$ bytes and $R = 1$ Mb/s. These results can be related to Figures 8 and 9, respectively, since they have the same general behavior, except for the fact that the range of traffic-load values over which the throughput is non-zero is smaller by more than an order of magnitude across all protocols, and there is a slight decrease in the maximum throughput values due to the small packet size. The cases for $L = 100$ bytes and $R = 300$ Mb/s are not shown due to lack of space, but all protocols have the same general behavior shown in Figures 10 and 11, and perform poorly due to the high τ/T ratio.

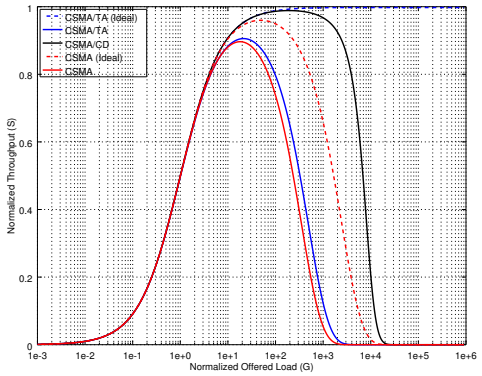


Fig. 12. S vs. G for $L = 100$ bytes, $R = 1$ Mb/s, and $r = 100$ m.

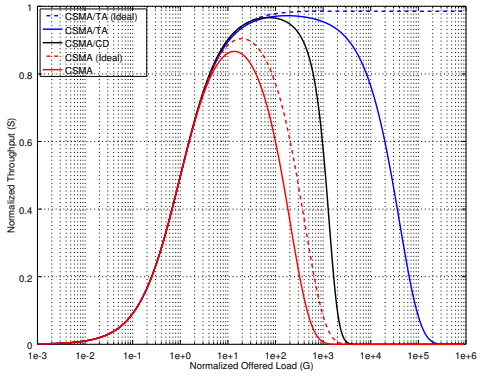


Fig. 13. S vs. G for $L = 100$ bytes, $R = 1$ Mb/s, and $r = 594.06$ m.

VI. Conclusions

We introduced Carrier-Sense Multiple Access with Transmission Acquisition (CSMA/TA) as an extension of CSMA for stations using half-duplex radios with a single antenna. CSMA/TA seeks to increase the likelihood of having a successful transmitting station among a group of colliding stations. It was shown that CSMA/TA can perform better than CSMA and CSMA/CD (which would require using full-duplex radios in WLANs) if the radio's turnaround times are close to the propagation delay. This is a very promising result, because the chipsets available in the market today and in the near future are such that turnaround times are being reduced dramatically. Given that half-duplex radios with much faster turnaround times are much cheaper than full-duplex radios, this makes CSMA/TA an attractive approach for future WLANs compared to traditional CSMA. Our future work addresses the embedding of CSMA/TA as part of the IEEE 802.11 standard for WLANs.

Acknowledgment

This work was supported in part by the Jack Baskin Chair of Computer Engineering at UCSC and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

References

[1] IEEE Std 802.11ah-2016 *(Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016)*, pp. 1–594, May 2017.

[2] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics," *IEEE Trans. Commun.*, 1975.

[3] R. M. Metcalfe and D. R. Boggs, "ETHERNET: Distributed packet switching for local computer networks," *CACM*, vol. 19, no. 7, pp. 395 – 403, 1976.

[4] S. Sen, R. R. Choudhury, and S. Nelakuditi, "CSMA/CN: Carrier sense multiple access with collision notification," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 544–556, Apr. 2012.

[5] J. Peng, L. Cheng, and B. Sikdar, "A wireless MAC protocol with collision detection," *IEEE Transactions on Mobile Computing*, vol. 6, no. 12, pp. 1357–1369, Dec 2007.

[6] L. Song, Y. Liao, K. Bian, L. Song, and Z. Han, "Cross-layer protocol design for CSMA/CD in full-duplex WiFi networks," *IEEE Communications Letters*, vol. 20, no. 4, pp. 792–795, April 2016.

[7] T. Vermeulen, F. Rosas, M. Verhelst, and S. Pollin, "Performance analysis of in-band full duplex collision and interference detection in dense networks," in *IEEE CCNC*, Jan 2016, pp. 595–601.

[8] *2.4GHz to 2.5GHz 802.11g/b RF Transceivers with Integrated PA*, MAXIM, 2011.

[9] *A True System-on-Chip Solution for 2.4GHz IEEE 802.15.4 and ZigBee Applications*, Texas Instruments, 2011.

[10] J. I. Choi, M. Jain, K. Srinivasan, P. Levis, and S. Katti, "Achieving single channel, full duplex wireless communication," in *ACM Mobicom*. ACM, 2010, pp. 1–12.

[11] C. L. Fullmer and J.J. Garcia-Luna-Aceves, "FAMA-PJ: a channel access protocol for wireless LANs," *Proc. ACM MobiCom '95*, 1995.

[12] R. Rom, "Collision Detection in Radio Channels," *Local Area and Multiple Access Networks*, Computer Science Press, 1986.

[13] R. Jurdak et al., "A survey, classification and comparative analysis of medium access control protocols for ad hoc networks," *IEEE Communications Surveys & Tutorials*, 2004.

[14] J.J. Garcia-Luna-Aceves, "Carrier-Sense Multiple Access with Collision Avoidance and Detection," *Proc. ACM MSWiM*, 2017.

[15] B. Hajek, "Random Processes for Engineers" Cambridge University Press, 2015.

[16] J.J. Garcia-Luna-Aceves, "Carrier Resolution Multiple Access," *Proc. ACM PE-WASUN*, 2017.

Prescriptive Analytics for MEC Orchestration

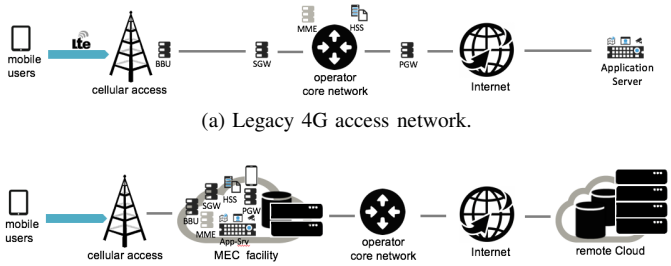
Alberto Ceselli*, Marco Fiore†, Angelo Furno‡, Marco Premoli*, Stefano Secci§, Razvan Stanica¶
* Università Degli Studi di Milano, Dept. of Computer Science, Crema, Italy. Email: {firstname.lastname}@unimi.it
† CNR–IEIIT, Torino, Italy. Email: marco.fiore@ieiit.cnr.it
‡ Univ Lyon, IFSTTAR, ENTPE, LICIT UMR_T9401, F-69675, Lyon, France. Email: angelo.furno@ifsttar.fr
§ Sorbonne Université, CNRS, LIP6, F-75005 Paris, France. Email: stefano.secci@sorbonne-universite.fr
¶ Univ Lyon, INSA Lyon, Inria, CITI, F-69621, Villeurbanne, France Email: razvan.stanica@insa-lyon.fr

***Abstract*—Orchestrating network and computing resources in Mobile Edge Computing (MEC) is an important item in the networking research agenda. In this paper, we propose a novel algorithmic approach to solve the problem of dynamically assigning base stations to MEC facilities, while taking into consideration multiple time-periods, and computing load switching and access latency costs. In particular, leveraging on an existing state of the art on mobile data analytics, we propose a methodology to integrate arbitrary time-period aggregation methods into a network optimization framework. We notably apply simple consecutive time period aggregation and agglomerative hierarchical clustering. Even if the aggregation and optimization methods represent techniques which are different in nature, and whose aim is partially overlapping, we show that they can be integrated in an efficient way. By simulation on real mobile cellular datasets, we show that, thanks to the clustering, we can scale with the number of time-periods considered, that our approach largely outperforms the case without time-period aggregations in terms of MEC access latency, and at which extent the use of clustering and time aggregation affects computing time and solution quality.**

I. Introduction

The softwarization of networks is an innovative trend expected to transform the mobile access environment in the coming years. It is an evolution accompanied by the virtualization of network functions and application servers, which can be operated running virtualization clusters close to, or at cellular base stations and mobile network points of presence [1]. The type of functions that can be virtualized ranges from traffic load balancers and multimedia (de)coders to mobile core functions such as those of the Long Term Evolution (LTE) Evolved Packet Core (EPC) [2]. Application servers can also be run in such facilities, so that the end-to-end user experience benefits from low access latency [3].

An illustration of this evolution is given in Fig. 1. Fig. 1a depicts a legacy 4G environment, where the user accesses remote applications via cellular access, in such a way that its wireless signals are processed at Base Band Unit (BBU) nodes integrated to cellular Base Stations (BSs), its traffic is routed through the EPC (composed of four main functions), before reaching the Internet border on the way to the application server. Fig. 1b shows instead a fully cloudified environment, where radio-network elements such as the BBU, EPC functions, mobile phone remotely executable applications, as well



(a) Legacy 4G access network.

(b) Fully cloudified access network.

Fig. 1. Mobile access network evolution with edge computing.

as application servers (possibly synchronized with a remote cloud) are all virtualized in potentially the same place, called Mobile Edge Computing (MEC) facility. Such a scenario is an extreme one, coping with the virtualization of a variate set of hardware, but that could correspond to the reality in the coming decade. In any case, the virtualization of a subset of these functions is a certainty, as demonstrated by different ongoing projects in the industry, for instance those regarding the virtualization of EPC functions (as announced by Orange Spain in fall 2017), of radio-access network elements (as announced by China Mobile in 2011), or of application servers (as encompassed in some reference MEC use-cases [1]).

Among the virtualizable nodes at MEC facilities, we can distinguish nodes that are strictly serving a subset of the BSs of an operator (e.g., vBBU and vEPC nodes), and nodes that serve single or multiple users (e.g., virtualized mobile device environment for computation offloading, virtualized application servers), possibly behind different BSs. An important amount of traffic can therefore be aggregated at MEC facilities, depending on the type of virtualized functions that are run at these edge delivery points. The management of virtualized nodes running at MEC facilities encompasses service and network management operations mainly related to: *i)* BS-to-MEC facility association, and *ii)* user-to-virtual machine (VM) association (a VM being in turn associated to a MEC facility). This kind of association decisions imply the execution of VM-level MEC orchestration operations, such as VM scaling up/down (increase/decrease of computing resources such as memory, processor, storage), Virtual Network Function (VNF) scaling in/out (more or less VM instances running a given VNF), VM migration, VM creation or destruction.

At the time being, the telco industry is more focused on the virtualization of the nodes that serve a subset of cellular

ISBN 978-3-903176-08-9

antennas (e.g., vBBU, vEPC), instead of working directly at the user-VM granularity, mainly because of scalability concerns. Therefore, one shall consider BS-to-MEC facility switching decisions as critical ones. In this paper, we indeed propose a MEC orchestration framework that primarily optimizes BS-to-MEC facility association over time, based on a spatiotemporal grouping of the BSs, while integrating VM workload adaptations across MEC facilities.

BS-to-MEC facility switching operations can not be reasonably expected to run continuously, as this would incur in traffic loss and overhead due to traffic handover, but to occur only at certain points in time (*e.g.*, once every thirty minutes). Hence, introducing an implicit time discretization of the orchestration system appears appropriate. In order to identify suitable discrete-time profiles of the traffic demand, different strategies can be employed. The simplest option is to aggregate the demand observed at each BS during every time step in a recent reference period, exploiting training data, using one reference profile for each time step. Another option to identify suitable discrete-time profiles of the traffic demand is to use temporal clustering analytics on the historical data, so as to group together time slots that feature very similar distributions of the mobile traffic demand across the BSs.

In this paper, we explore the two options above, proposing a prescriptive analytics approach integrating advanced temporal clustering into a mathematical programming formulation of the addressed MEC orchestration problem. The clustering returns a limited number of profiles, each of which corresponds to time intervals where the mobile network presents a similar distribution of the demand. It is then possible to feed the optimization framework with a small number of profiles, with the risk of decreasing the solution quality, since typical profiles can only approximate the actual MEC network load at a specific time step. We assess in the paper the computational and quality aspects of our prescriptive analytics approach, as compared to basic time aggregation in the orchestration.

The manuscript is structured as follows. Section II draws the necessary background. Our network model is described in Section III, while our orchestration algorithm is described in Section IV. Section V reports numerical results. Section VI concludes the paper.

II. Background

We draw in this section the necessary background on edge computing and virtualization and on the integration of data analytics in network optimization problems.

A. Edge computing and network virtualization

In a MEC infrastructure, virtualization clusters – called 'MEC facilities' or 'MEC hosts' in the standardization documents [1], or 'cloudlets' in academic jargon [5] – are connected to access network nodes within a few hops, to deliver access to application servers running as VMs. Various operations dealing with the changing mobile access demand can be applied to orchestrate the resulting cloud-network system, which include BS to MEC facility dynamic assignment, VM capacity rescaling (addition or removal of computing power in terms of live memory or virtual processors) and VM migration (a VM state is moved from one MEC facility to another one). An 'orchestrator' is in charge of instantiating such decisions into the MEC infrastructure. Each orchestration action comes at a cost, often referred to as 'migration' or 'switching' cost, as it can require synchronizing states and reconfiguring network equipment and servers, across a geographical network under stringent performance guarantees. The technology to perform MEC orchestration operations is being experimented since many years [6]. It commonly takes into consideration changing states of the network in time and space, related to user mobility and digital usages behavior.

These dynamics are being considered for the management of not only application servers, but also of the network services needed to deliver resilient access to applications. Indeed, 5G systems will also build on new networking paradigms such as Network Function Virtualization (NFV) and Software Defined Networking (SDN) in order to, on the one hand, support the orchestration of virtualized network functions and, on the other hand, provide to core network switches the necessary features to support flow management that may be needed when applying fine-grained orchestration decisions [7].

Eventually, for mobile access networks and in particular cellular networks, the physical facility delivering application and network function VMs is expected to be the same, as already discussed in Fig. 1, located in access network aggregation points of presence. Such a convergence is also clearly appearing in standardization efforts related to MEC and NFV systems [8], [9], with equivalent interfaces between virtualization layer and orchestration system components.

A significant amount of work exists in the area of MEC and mobile-access NFV orchestration. A common problem addressed is the virtualization cluster placement within the access network, as considered in [10] for application VMs, in [4] for the EPC functions and in [11] for radio-access functions. A different orchestration dimension is the one related to VM migration and rescaling across a given set of MEC facilities, as a function of user mobility, as addressed in [12] for application VMs and in [13] for the EPC functions. Finally, in the area of virtualized radio-access network orchestration, the problem of clustering, i.e., assigning a set of BSs to BBUs was also extensively studied, as for instance in [14].

B. Data-driven mobile networks

A further step in this area, only marginally addressed to date, is to investigate how to integrate the result of data analytics in the instrumentation of MEC orchestration decisions, related to placement, migration, rescaling and clustering operations, along the lines traced in [15], [16].

Virtualized networks where significant resources are placed in proximity of the radio access open substantial new scenarios for the dynamic management of system operations. Solutions based on data analytics are in particular expected to play a critical role: knowledge inferred by mining traffic measurements and Key Performance Indicators (KPIs) will fuel effective

orchestration policies for the deployment and re-allocation of resources across mobile edge computing facilities. The vision of 'data-driven' (also referred to as 'cognitive') network management is attracting the interest of a growing research community [17], [18], and is supported by major players in the 5G ecosystem [19].

Due to the very recent emergence of relevant use cases, solutions to extract useful information from massive amounts of mobile traffic data records and to employ it for network configuration are still in their infancy. Data analytics for mobile network traffic based on clustering or spectral analysis have revealed regular macroscopic structures [20], [21] that are highly predictable [22]. Actual experiments of data-driven network management have mainly focused on optimizing video streaming services [23], [24] and controlling core network congestion [25], [26]. However, as of today there is almost no practical demonstration of how MEC can benefit from data-driven paradigms. The single application we could identify is the data-driven BBU-to-BS clustering approach in [27], where however the interconnection network is not modeled.

In this paper, we present a first application of data-driven networking in the context of MEC orchestration, and more precisely clustering decisions, considering both network and systems constraints. Specifically, we leverage existing analytics for the spatiotemporal classification of traffic, and extract long-timescale patterns in the spatial distribution of the mobile traffic demand. We then employ these patterns to guide the operation of MEC facilities so that the user Quality of Service is maximized, by their integration in orchestration algorithms based on mathematical programming.

C. Network optimization

The orchestration problem we address is to find groups of BSs for their association to MEC facilities, in a multi-period setting such that the BS-to-MEC facility association can change across periods. In the area of network optimization, this requires to tackle a multi-period extension of the famous Generalized Assignment Problem (GAP) [28].

We point to [29] for a detailed review on the GAP and its extensions. Despite the large body of research available on the GAP, we are not aware of many papers directly dealing with its multi-period extensions. In [30], the authors face a single-source allocation problem with a flexible model and an effective algorithm; however, their model does not handle limited capacity, which is a crucial feature in our application. The multi-period allocation problem discussed in [31], in which a dual ascent technique is adapted to telecommunication networks applications, is similarly missing the handling of capacities.

Although our problem does not require to decide the location of the facilities, which is instead assumed to be optimized in a prior strategic planning [10] and given as input, one may expect features and computational challenges similar to those of multi-period location problems [33]. Recent approaches on that field include [34]: the authors face a multi-period concentrator location and dimensioning problem, providing MILP formulations and reduction techniques, and solving to optimality in less than one hour of computation instances with up to 30 clients, 10 candidate location sites and 15 time periods, or 100 clients, 30 candidate locations and 5 time periods. In [35] the authors introduce exact methods for a capacitated multi-period facility location problem in which, however, unlike our case, the demand of each client can be fractionally served by multiple facilities. Large scale instances with up to 200 facilities, three periods and an arbitrary number of clients could be solved with their algorithms.

III. MEC NETWORK ORCHESTRATION MODEL

We elaborate our reference MEC network orchestration model along the following generic lines. BSs have associated mobile traffic demand, that changes over time. Each MEC facility has a certain capacity, limiting the overall amount of demand it can serve simultaneously. BSs must be assigned to MEC facilities; each new assignment implies a cost for each user connected to the BS in terms of latency for communicating with the associated MEC server. Due to capacity limits, it might not always be a good decision to assign each AP to its MEC facility of minimum latency; furthermore, since demand changes over time, an assignment pattern would hardly remain an efficient one over the whole planning horizon. We therefore leave the option of changing assignments over time, taking into account that each change implies a switching cost for the network, for example in terms of signaling to move session data of active users. An optimization problem therefore arises, that is to assign BSs to MEC facilities over time, respecting capacity constraints and minimizing a combination of users (assignment) and network (switching) costs.

Before providing a more formal problem statement and mathematical formulation, we describe the data analytics problem we address to instrument the orchestration algorithm.

A. Data analytics

The data analytics we adopt to drive our resource orchestration problem is inspired by the temporal classifier of mobile network traffic introduced by [37]. The classifier leverages an agglomerative hierarchical clustering with fine-tuned distance measures, and allows detecting long time periods during which the geographic distribution of the mobile traffic demand does not vary significantly. The results presented in the original paper show that, *e.g.*, the aggregate demand of voice calls and text messages switches among a very small number of possible spatial configurations during a whole week.

We employ the classifier above as a building block, and proceed through the 4 phases, also summarized in Fig. 2:

Phase 1: We collect substantial measurement data from an operational mobile network. The data captures the demand for a major mobile service in two large-scale metropolitan regions for a period of several consecutive months. Details on the data collection are provided in Sec. V-A.

Phase 2: For a subset of the collected data, representing our training set, we compute the typical weekly average demand, by aggregating all data collected at the same time of the week.

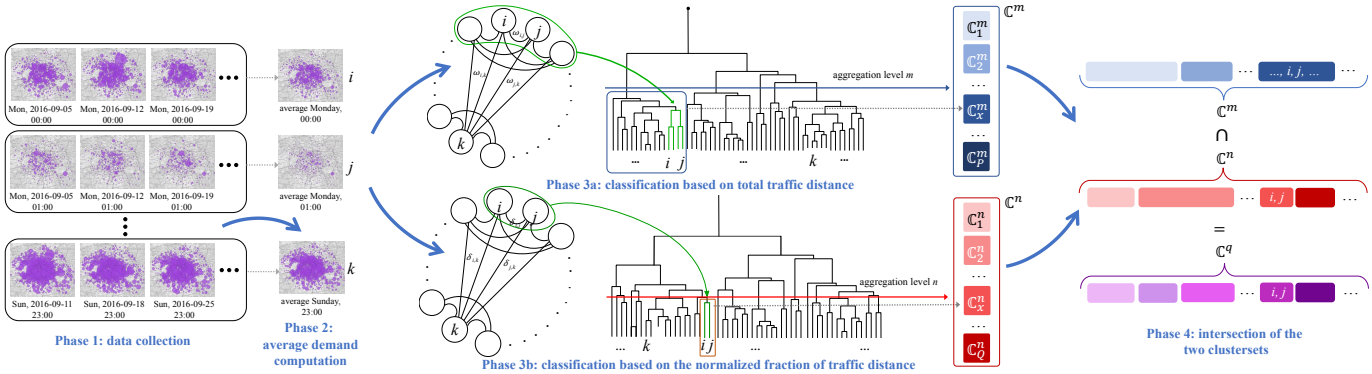


Fig. 2. Workflow for the classification of network usage profiles. The final intersection cluster set $\mathbb{C}_q$ is used for the training.

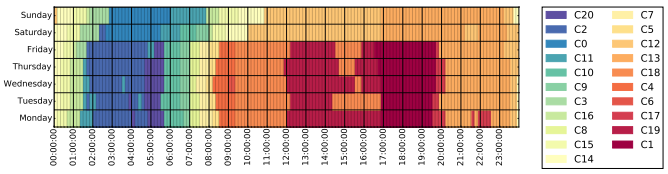


Fig. 3. Sample cluster set of 10-minute time instants for a mobile service. The plot outlines the existence of 20 temporal classes of spatial distributions of the demand, during each daytime (abscissa) of different weekdays (ordinate).

For instance, the representative offered load on Monday, 4:00 pm at one antenna is the average of all measurements on Mondays in the training set, at that specific time and antenna. Clearly, time needs to be discretized in order to obtain a finite set of time instants: we thus assume that each time instant refers in fact to a period of duration T.

Phase 3: Following the methodology suggested in [37], we run two separate instances of the classifier on the average week, considering two distance metrics to compute the similarity of demands at diverse time instants. The first such metric is the difference of total traffic volumes, which tends to cluster together time instants with equivalent total demands. The second metric is the difference of the normalized fraction of traffic at each antenna, which groups together time instants that feature comparable spatial distributions of the demand.

Phase 4: We derive the intersection of the two cluster sets, obtaining our final set of time instant classes. The rationale is that such an intersection yields classes that have *i)* equivalent total traffic volumes, that are *ii)* distributed in the same way across antennas. In other words, the demands in time instants that belong to the same class are similar from all viewpoints.

Fig. 3 shows an example of the final cluster set for our reference mobile service, when considering that each time instant spans $T = 10$ minutes. In the specific case under study, our approach categorizes all 10-minute time periods in a week into just 20 classes, *i.e.*, spatial configurations of the mobile service demand. The fact that 20 classes capture the diversity of offered loads in more than 1,000 time instants underscores how the demand for our target mobile service shows significant regularity over time. The emergence of 20 classes is also good news for our case study, as it implies that a small number of MEC facility deployments can be sufficient to accommodate all possible spatial dynamics in the traffic.

Another interesting observation is that time instants in a same class are typically contiguous. Also this aspect plays in favor of our objective: the temporal consistence of spatial configurations entails that MEC resource allocation profiles remain valid throughout quite long timespans, and the number of switches between profiles is reduced. The expectation (confirmed by our numerical evaluation) is that these intrinsic properties of the mobile service demand can make a data-driven approach for MEC deployment highly effective.

B. Orchestration Optimization Model

Our MEC orchestrator includes an optimization core for performing prescriptive analytics on a tactical level. We adapt models and methods from [38]. In particular, we build dynamic assignment plans detailing, for each time slot, the set of BSs to be connected to each MEC facility and, as a by-product, the set of switching operations to be performed between subsequent time slots. We consider a periodic single-assignment operational policy, that is, in each time slot each BS is assigned to exactly one MEC facility, and the last time slot is assumed to be followed by the first one.

The task details are the following.

Input. We assume to be given the set of BSs, the set of MEC facilities and a discretization of the time horizon in a set T of time slots. We also assume to be given *i)* for each BS, the mobile traffic demand that has to be accommodated in each time slot, *ii)* the capacity of each MEC facility, *iii)* the physical distance between each BS and each MEC facility and the network distance between each pair of MEC facilities (that is, a measure directly proportional to the network latency, including packet processing latency at intermediate nodes, and physical distance).

Output. We expect, as output of the optimization core, an assignment plan: for each BS and each time slot, an indication of the MEC facility where traffic needs to be routed. As a side result, we expect a switching plan, that is a boolean value for each BS and each pair of MEC facilities for each time slot, indicating whether that BS switches at that time between a

particular pair of MEC facilities, or not.

Requirements. The assignment plan satisfies the following conditions: *i)* the overall demand assigned to each MEC facility at each time slot must not exceed its capacity, *ii)* each BS is connected to exactly one MEC facility at each time slot, *iii)* assignment and switching plans must be coherent.

Objective. The plans must target a trade-off between the minimization of network- and user-related costs. The former is generated by the change of BS-MEC facility associations in consecutive time slots, which produces some overhead due to the necessity of migrating VMs. The latter is instead the latency experienced by the user with the current BS-MEC facility association. The relative weight of the network- and user-related costs in the objective function is represented by suitable parameters, set to equal weights in our experiments.

A sample instance with three APs (squares), two MEC facilities (circles) and two time-slots (left and right parts) is depicted in Fig. 4: AP 2 is assigned to MEC facility A at $t = 1$ and MEC facility B at time $t = 2$, therefore a switching operation from A to B needs to be performed.

Formally, our orchestration task can be modeled with the following Mathematical Program:

$$\min \alpha \sum_{t\in T}\sum_{i\in A}\sum_{\substack{(j,k)\in \\ K\times K}} d_i^t l_{jk} y_{ijk}^t \quad +\beta \sum_{t\in T}\sum_{i\in A}\sum_{k\in K} d_i^t m_{ik} x_{ik}^t \tag{1}$$

$$\text{s.t.} \sum_{i\in A} d_i^t x_{ik}^t \leq C_k \qquad \forall t \in T, \forall k \in K \tag{2}$$

$$\sum_{k\in K} x_{ik}^t = 1 \qquad \forall i \in A, \forall t \in T \tag{3}$$

$$x_{ik}^t = \sum_{l\in K} y_{ilk}^t \qquad \substack{\forall i\in A,\forall k\in K \\ \forall t\in T\setminus\{1\}} \tag{4}$$

$$x_{ik}^t = \sum_{l\in K} y_{ikl}^{t+1} \qquad \substack{\forall i\in A,\forall k\in K \\ \forall t\in T\setminus\{T\}} \tag{5}$$

$$x_{i,k}^t \in \{0,1\}^{\substack{\forall i\in A,\forall k\in K \\ \forall t\in T}}, \qquad y_{i,k',k''}^t \in \{0,1\}^{\substack{\forall i\in A,\forall t\in T \\ \forall k',k''\in K}} \tag{6}$$

where A is the set of BSs, K is the set of MEC facilities, T is the set of time slots, d_i^t is the demand of BS $i \in A$ during time slot $t \in T$, C_k is the capacity of MEC facility $k \in K$ and l_{jk} (resp. m_{ik}) is the distance from facility j to facility k (resp. from BS i to facility k). Assignment plans are encoded by variables x_{ik}^t, which take value 1 if BS i is assigned to facility k at time t, 0 otherwise. Switching plans are encoded by variables y_{ijk}^t, which are 1 if traffic from BS i must be switched from facility j to facility k at time t, 0 otherwise. Constraints (2) and (3) model requirement *i)* and *ii)*, respectively. Collectively, constraints (4) and (5) ensure assignment and switching plans to be coherent. Finally, (6) impose binary value to all variables. The first term in the objective function (1) models switching costs, while the second term models assignment costs.

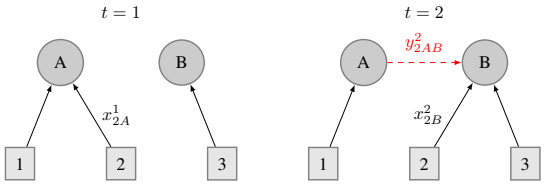


Fig. 4. x and y variables

IV. Resolution Algorithm

Problem (1) – (6) is NP-Hard in general [29]. When the size of the MEC network is large, the resolution of the orchestration problem requires ad-hoc algorithms. A decomposition approach can be employed: when the requirements on MEC facility capacities are considered in 'soft' form, penalizing their violation with suitable multipliers in the objective function, the problem disaggregates in independent subproblems per BS.

In particular, when a penalty multiplier is fixed for each capacity unit violation, a minimum cost path problem in a suitable graph needs to be solved independently for each BS; these problems admit polynomial time algorithms, and can therefore be computed efficiently also on large scale networks and fine grained time discretizations.

In turn, we are able to obtain a set of optimal multipliers through Dantzig Wolfe Reformulation and Column Generation methods [38]. Strictly speaking, the solution found by means of Dantzig Wolfe Reformulation might be fractional, that is some BSs might be fractionally assigned to more than a single MEC facility in some time slots. In that case, we run a heuristic, selectively rounding these fractions, and thereby always choosing a single MEC facility. Our heuristic works as follows. We iterate over each time slot in sequential order. In each time slot, for each BS, the MEC facility having highest fractional assignment is retrieved: we sort the set of BSs according to these assignment values (from highest to lowest). Then, we iterate over each BS following this order, assigning a single MEC facility to each BS; whenever an integer assignment would yield a capacity violation, an alternative MEC facility is selected for that BS, still in order of non-increasing fractional assignment values in the particular time slot. Our computational experiments revealed this heuristic to be highly effective.

Furthermore, we employ several techniques for speeding up our algorithms, like the use of Lagrangean fixing procedures to reduce the search space. In particular, we initialize column generation using greedy heuristics: for each time slot, BSs are sorted by non-increasing demand and each BS is associated to a profitable MEC facility. The most profitable MEC facility is considered to be that involving no switching cost, if enough residual capacity is available, or the nearest one with enough residual capacity, otherwise.

V. Simulation results

We implemented our algorithms in C++, using CPLEX 12.6 [39] to solve the master LP subproblems, running tests on an Intel i7 4GHz workstation equipped with 32 GB of RAM. Before describing the results on orchestration time period assessment and MEC performance, we describe the dataset.

A. Dataset

The dataset used in our study was collected in the core network of Orange, a major European mobile operator, during three months in 2016. It describes the traffic generated by several millions of mobile subscribers in the French metropolitan areas of Lyon and Paris, for a specific mobile service, *i.e.*, Facebook. More precisely, the data was recorded by monitoring IP sessions at the 3G and 4G core network gateways. A combination of Deep Packet Inspection (DPI) and proprietary fingerprinting tools was employed to infer application-level information on Facebook user sessions. The approach allows to determine the volume of all content traffic related to the Facebook mobile service, including streaming content or messaging, accessed through the app or web interfaces.

The rationale for the choice of Facebook is that it represents a prominent mobile service, generating around 20% of the compound downlink and uplink demand in the network. It is also an example of typical service that could benefit from the improved quality of service granted by a MEC infrastructure.

It is important to remark that the traffic is aggregated at the antenna sector level. This ensures that the information in the dataset is a combination of the Facebook sessions of many users, hence it does not contain personal data or raises privacy issues. The dataset is composed of twelve weeks of traffic demands, aggregated by 10-minutes time-periods.

B. Time-period granularity

In order to identify suitable demand discrete-time profiles, we evaluated six different time-period aggregations:

- aggregating consecutive 10-minute periods to form a period of four, two and one hour ('4H', '2H' and '1H' in the remainder). The resulting training sets consist of 42, 84 and 168 time-periods, respectively. Using shorter time periods for this strategy revealed to be too complex to solve; at the same time we found tests on longer time periods not informative, as already the 4H case is dominated by 2H and 1H aggregations, and the experimental trend is clear, as discussed in the remainder;
- aggregating 10-minute periods belonging to the same clustering profile generated as presented in Subsection III-A, with clustersets of one hour ('1HC'), 30-minute ('30MC') and 10-minute ('10MC'); the resulting training sets differs for Lyon and Paris dataset: for Lyon dataset, time-periods are 99 for '1HC', 171 for '30MC' and 260 for '10MC'; for Paris dataset, time-periods are 60 for '1HC', 160 for '30MC' and 141 for '10MC'.

We observe that the number or time-periods in 2H is similar to that of 1HC. The same can be observed for 1H and 30MC.

For the training set, we use the first 4 weeks of the dataset to build the typical average week used by the clustering approach in the classification process. As recommended in [37], we also tested the construction of the typical week using the median value of the demand, but the results only marginally differed with respect to the average week, so we avoid reporting them.

Lyon dataset contains demands from 332 BSs, while Paris dataset has 1907 BSs. We set three cardinalities of facilities for the dataset of Lyon (10, 20 and 30, resp.) and two cardinalities of facilities for the dataset of Paris (20 and 50, resp.). The location of the facilities was generated by a k-medoid algorithm, using the coordinates of the BS locations as input data. Distances between BS and facilities were computed using the Haversine formula [40]. Parameters α and β of objective function (1) were both set to 1. The resulting training set is composed of 60 instances.

a) Benchmark: As benchmark for our methodology, we considered a baseline approach without the time-period aggregation we propose with our model, therefore with a single time-period, leading to a single assignment for every BS to a MEC facility over the week and no switching of assignment among MEC facilities during the week. We computed a single time-period demand averaging demands of all time-periods in our dataset; we used this single-time average demand to train our model for every city and every facility cardinality (5 training instances). We solved the corresponding problem with the ILP general solver of CPLEX, stopping the resolution when an optimality gap lower than 1% was reached. We label such instances as 'S' in the remainder.

b) Training Computational Results: In Fig. 5 we present the box-plots of the execution times of the training sets, in logarithmic scale (base 10), highlighting each the different time-period granularity (a boxplot shows a box bars indicating the minimum, 1st quartile, median, 3rd quartile, maximum). We can notice that the consecutive 1H case has the highest execution time (up to 10 hours of executions), followed by the 30MC case, while a lower time is required by 4H and 2H and 1HC, with S as fastest approach. In addition to the plot, we found that the average execution time for 'S' is 10 seconds, for '4H', '2H' and '1HC' is less than five minutes (165.1 s, 273.5 s and 268.8 s, resp.), for '1H' is more than 2 hours (9022.6 s), for '30MC' is slightly more then 1 hour (4515.6 s) and finally for '10MC' half an hour (2012.9 s). Having similar number of time-periods, '2H' and '1HC' (resp. '1H' and '30MC') require similar training time.

In Fig. 6 we present the box-plots of the optimality gap of the training sets, still highlighting each the different time-period granularity. As specified previously, 'S' training was stopped as soon as an optimality gap less than 1% was reached. We can notice little difference among different aggregations: with respect to the worst-case performance, '4HC' has the worst result, while the clustering cases have better performances; with respect to the median performance, all aggregations show similar results in the range 2–5%. In addition to the plot, we found that the average optimality gap for '4H' is around 5%, for '2H', '1H' and '1HC' is around

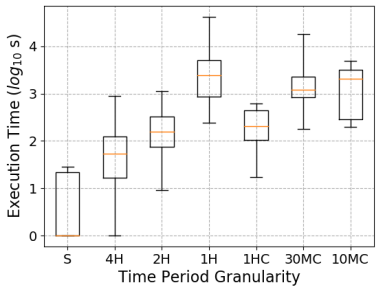


Fig. 5. Execution Time

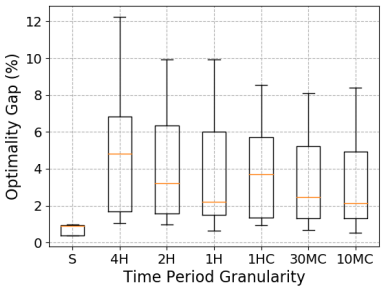


Fig. 6. Optimality Gap

4%, while '30MC' and '10MC' is around 3.3%.

C. Cost components assessment

We tested the assignments generated by our algorithm against the original 10-minute periods demands in our datasets, considering all twelve weeks separately. That is, for every training instance we have twelve tests with a different demand.

First, we compare the performance of the time-periods aggregation from the point of view of the MEC access latency costs, that is defined by the objective function of model (1).

We present the costs in three parts:

- the assignment cost considering the distance between a BS and MEC facility and the demand of the BS (component $\beta \sum_{t\in T} \sum_{i\in A} \sum_{k\in K} d_i^t m_{ik} x_{ik}^t$ in (1));
- the switching cost considering the distance between MEC facilities in considering time-slots and the demand of the BS (component $\alpha \sum_{t\in T} \sum_{i\in A} \sum_{\substack{(j,k)\in \\ K\times K}} d_i^t l_{jk} y_{ijk}^t$ in (1));
- and the total cost (1).

We do not present the absolute value of the assignment and total costs, rather for every test week we compute the percentage difference between the lowest cost among those obtained with the seven time-period aggregations (4H, 2H, ... 10MC, plus the benchmark S) and the cost obtained with the given time-period aggregation. For example, while testing with 10 facilities, trained with average reference week, let us assume that for test week 1 the minimum cost $\bar{c}_1$ is given by the training assignment generated by '30MC' aggregation: the percentage difference of the costs of the test week 1 of every time-period aggregation is computed as $(c_1 - \bar{c}_1)/\bar{c}_1$. Hence, '30MC' will have a percentage gap of 0 for test week 1, in the scenario with 10 facilities trained with average week.

In Fig. 7 we present the box-plots of the percentage gaps of the costs. Every figure contains a separate box-plot for each time-period aggregation method. We can notice that:

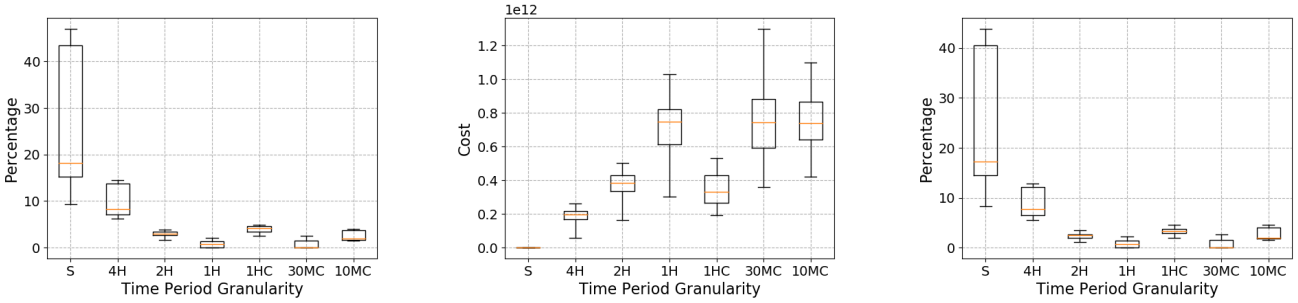
- the positive effect of clustering can be evaluated by comparing '1H' with '30MC': they yield a similar number of time-periods, but the latter allows slightly faster training, producing at the same time solutions of lower costs;
- w.r.t. assignment costs (Fig. 7a), 'S' always leads to the highest cost, i.e. longer MEC access latency, on median 20% higher than the minimum; all other aggregations show similar results, except '4H' which has slightly worse results (on median 8% higher cost than the minimum): the lowest average cost is given by '30MC', '1H' retrieves a cost higher on median of 1%, while '2H','1HC' and '10MC' show worsening of about 2-4%;
- on the contrary, w.r.t. switching costs (Fig. 7b), with the exception of 'S' which always ensures no switching costs, '4H' always leads to the lowest cost, i.e., the least number of MEC facility switching, and on median all other aggregations retrieve a switching costs from 2 to 8 times the cost given by '4H';
- however, the huge difference in the switching cost does not lead to a significant change in the total cost (Fig. 7c): this latter is composed mostly of the aggregation costs and it shows similar difference gaps.

In order to further analyze this behavior, in Fig. 8 and 9 we present two indices regarding the assignments and the switching arising from the training:

- given that every BS has to be assigned to a MEC facility in every time-period, in our model the best option corresponds to the nearest facility. Therefore, we compute for every case the percentage of times a BS has not been assigned to its nearest MEC facility, that we present in Fig. 8 as single box-plots for every time-period aggregation. We can notice that 'S' has the highest median non-nearest assignments (more than 30% of the assignments); '4H' and '1HC' have similar median behavior with 26% of non-nearest assignments, and all other aggregations show a value around 22%. This behavior better explains the poor performance of 'S' and '4H' for what concerns assignment costs.
- in Fig. 9 we present the percentage of times a switching occurs in any time-period for any BS (i.e. number of switching over $(|T| - 1) \cdot |A|$), as box-plot for each time-period aggregation: we can notice that this value is considerably low for '4H' (on average less than 0.5% of the time a switching occurs) and is on average low for every aggregation (the highest value is 1.25% of the times). This behavior also explains the low effect of the switching costs in the total cost computation.

D. Computing capacity violation

Given that the BS-MEC facility assignment is computed using a reference week, it can generate a violation of the MEC facility capacity given by the change of the demand pattern in



(a) Assignment Costs (b) Switching Costs (c) Total Costs

Fig. 7. MEC access latency and switching costs gaps.

the test week from the reference week. In order to measure the violation of capacity, we introduce three indices:

- average capacity excess ('SUM-SUM' in the remainder):

$$\sum_{t\in T}\sum_{k\in K}\frac{\max\{\sum_{i\in A} d_i^t x_{ik}^t - C_k, 0\}}{C_k\cdot|K|\cdot|T|}$$

- percentage number of times a capacity is exceeded ('SUPPORT' in the remainder):

$$\frac{|\{(t,k):\sum_{i\in A} d_i^t x_{ik}^t - C_k < 0, \forall t\in T, \forall k\in K\}|}{|K|\cdot|T|}$$

- average of excess, only when a violation occurs ('SUM-SUM-SUPPORT'):

$$\frac{\sum_{t\in T,k\in K:d_i^t x_{ik}^t - C_k<0}\left(\frac{\sum_{i\in A} d_i^t x_{ik}^t - C_k}{C_k}\right)}{|\{(t,k):\sum_{i\in A} d_i^t x_{ik}^t - C_k < 0, \forall t\in T, \forall k\in K\}|}$$

In Fig. 10 we present box-plots of these three indices, in logarithmic scale (base 10). We can notice that:

- the 'SUM-SUM' index (Fig. 10a) is rather low for every time-period aggregation, only the '10MC' show a slightly higher median value, but it is less than the 0.05% for both the reference weeks;
- the 'SUPPORT' index (Fig. 10b), i.e. the percentage of time-periods in which a MEC facility has a capacity violation, does not show particular differences between the time-periods aggregation; '4H' shows a lower third quartile, that is however always lower than 0.5% for every time-period aggregation;
- the 'SUM-SUM-SUPPORT' index (Fig. 10c), i.e. the average violation computed only when violations occur, show a different behaviour for the '10MC' aggregation: while the median value is almost constant for all aggregations, '10MC' in the worst-case can violates a MEC capacity of more than 350% (i.e. it assign to a facility an amount of demand that is more than three times its capacity); this behaviour would advise against '10MC'.

VI. Conclusions

We presented in this paper a MEC orchestration framework that (i) enables taking orchestration decisions on base station to MEC facility assignments, and that (ii) at an arbitrary time period granularity within a reference horizon hence taking into consideration load variations along time, while (iii) supporting advanced spatio-temporal clustering among base stations based on network data analytics. It is, as of our knowledge, the first effort of this type.

We show that - by extensive simulations against real network data of an application that could benefit from MEC - with our framework we (a) largely outperform baseline orchestration decision without time-period aggregation by a order of magnitude in terms of MEC access latency, (b) scale with the number of time periods by leveraging on spatio-temporal clustering of base stations, and (c) identify which time-periods and aggregation techniques better allow minimizing MEC access latency and facility switching costs.

As a further work we plan at refining the clustering algorithms so as to anticipate factors in the preprocessing phase that could enhance the quality of the orchestration solutions.

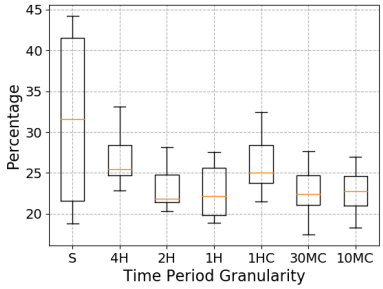


Fig. 8. Non-Nearest Assignments

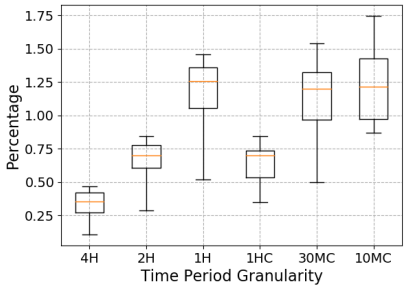
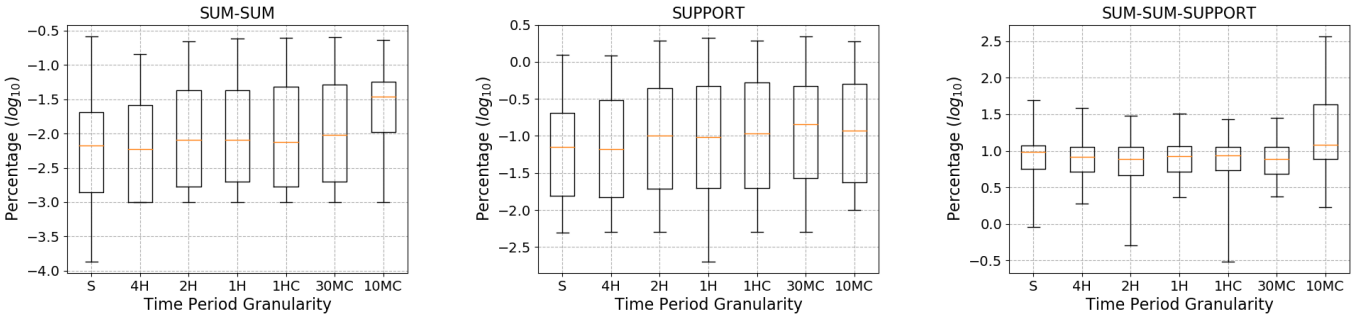


Fig. 9. Switching Occurrences



(a) SUM-SUM (b) SUPPORT (c) SUM-SUM-SUPPORT

Fig. 10. Capacity violation measures.

ACKNOWLEDGEMENTS

This work has been partially funded by "Piano di Sostegno alla Ricerca 2015-17" (Università degli Studi di Milano), the ANR ABCD (Grant No: ANR-13-INFR-005) and FP7 MobileCloud (Grant No. 612212) projects. We thank C. Ziemlicki from Orange for the support with data collection.

REFERENCES

[1] M. Patel et al., "Mobile-Edge computing introductory technical white paper", ETSI MEC, Tech. Rep., 2014.

[2] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)", *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[3] A. Aijaz et al., "Realizing the tactile internet: Haptic communications over next generation 5g cellular networks", *IEEE Wireless Communications*, vol. 24, no. 2, pp. 82–89, April 2017.

[4] D. Dietrich et al., "Network function placement on virtualized cellular cores", in *Proc. of COMSNETS 2017*.

[5] M. Satyanarayanan, "Cloudlets: at the leading edge of cloud-mobile convergence", in *Proc. of ACM QoSA 2013*.

[6] D. Lindemeier, "MEC proofs of concept," ETSI, [Online]: http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing/mec-poc

[7] P. Rost et al., "Mobile network architecture evolution toward 5G," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 84–91, May 2016.

[8] "Mobile Edge Computing (MEC); Framework and Reference Architecture", ETSI DGS/MEC-003Arch, V1.1.1 (2016-03).

[9] "Network Functions Virtualisation (NFV); Architectural Framework", ETSI RGS/NFV-002, V1.2.1 (2014-12).

[10] A. Ceselli, M. Premoli, S. Secci, "Mobile edge cloud network design optimization", *IEEE Transactions on Networking*, vol. 25, no. 3, 2017.

[11] C. Colman-Meixner et al., "Resilient cloud network mapping with virtualized BBU placement for cloud-RAN", in *Proc. of ANTS 2016*.

[12] S. Secci, P. Raad, P. Gallard, "Linking virtual machine mobility to user mobility", *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 927–940, Dec 2016.

[13] T. Taleb, A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks", *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[14] MY. Lyazidi, N. Aitsaadi, R. Langar, "Dynamic resource allocation for Cloud-RAN in LTE with real-time BBU/RRH assignment", in *Proc. of IEEE ICC 2016*.

[15] H. Assem, T. S. Buda, L. Xu, "Initial use cases, scenarios and requirements", *H2020 5G-PPP CogNet, Deliverable D2.1*, 2015.

[16] K. Zheng et al., "Big data-driven optimization for mobile networks toward 5G", *IEEE Network*, vol. 30, no. 1, pp. 44–51, January 2016.

[17] H. Yao et al., "A novel framework of data-driven networking", *IEEE Access*, vol. 4, pp. 9066–9072, 2016.

[18] J. Jiang, V. Sekar, I. Stoica, H. Zhang, *Unleashing the Potential of Data-Driven Networking*. Springer International Publishing, 2017.

[19] "EC H2020 5G infrastructure PPP pre-structuring model, version 2.0," 5GPPP, http://5g-ppp.eu, April 2014.

[20] A. Furno et al., "A tale of ten cities: Characterizing signatures of mobile traffic in urban areas", *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, October 2017.

[21] A. Furno, M. Fiore, R. Stanica, "Joint spatial and temporal classification of mobile traffic demands," in *Proc. of IEEE INFOCOM 2017*.

[22] J. Wang et al, "Spatiotemporal Modeling and Prediction in Cellular Networks: A Big Data Enabled Deep Learning Approach", in *Proc. of INFOCOM 2017*.

[23] A. Ganjam et al., "C3: Internet-scale control plane for video quality optimization", in *Proc. USENIX NSDI 2015*.

[24] J. Jiang et al., "CFA: A practical prediction system for video QoE optimization", in *Proc. of USENIX NSDI 2016*.

[25] K. Winstein, H. Balakrishnan, "TCP ex machina: Computer-generated congestion control", in *Proc. of ACM SIGCOMM 2013*.

[26] M. Dong et al., "PCC: Re-architecting congestion control for consistent high performance", in *Proc. of USENIX NSDI 2015*.

[27] L. Chen et al., "Complementary Base Station Clustering for Cost-Effective and Energy-Efficient Cloud-RAN", in *Proc. of UIC 2017*.

[28] S. Martello, P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[29] D. R. Morales, H. E. Romeijn, "The generalized assignment problem and extensions", in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds., 2005, pp. 259–311.

[30] R. Freling, H. E. Romeijn, D. R. Morales, A. P. M. Wagelmans, "A branch-and-price algorithm for the multiperiod single-sourcing problem", *Operations Research*, vol. 51, no. 6, pp. 922 – 939, 2003.

[31] I. Murthy, P. K. Seo, "A dual?ascent procedure for the file allocation and join site selection problem on a telecommunications network", *Networks*, vol. 33, no. 2, pp. 109 – 124, 3 1999.

[32] I. Murthy, "Solving the multiperiod assignment problem with start-up costs using dual ascent", *Naval Res. Log.*, vol. 40, pp. 325–344, 1993.

[33] S. Nickel, F. S. da Gama, "Multi-period facility location," in *Location Science*, G. Laporte, S. Nickel, and F. S. da Gama, Eds. Springer, 2015, pp. 289–310.

[34] I. Gourdin, O. Klopfenstein, "Multi-period capacitated location with modular equipments", *Comp. Op. Res.*, vol. 35, no. 3, pp. 661–682, 2008.

[35] J. Castro, S. Nasini, F. Saldanha-da Gama, "A cutting-plane approach for large-scale capacitated multi-period facility location using a specialized interior-point method", *Mathematical Programming*, vol. 163, no. 1, pp. 411–444, 2017.

[36] R. Halper, S. Raghavan, M. Sahin, "Local search heuristics for the mobile facility location problem", *Computers & Operations Research*, vol. 62, pp. 210 – 223, 2015. [Online].

[37] A. Furno, D. Naboulsi, R. Stanica, M. Fiore, "Mobile demand profiling for cellular cognitive networking", *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, March 2017.

[38] A. Ceselli, M. Fiore, M. Premoli, S. Secci, "Optimized Assignment Patterns in Mobile Edge Cloud Networks", *Comp. Op. Res*, 2018.

[39] *IBM ILOG CPLEX 12.6 User Manual*. IBM corp., 2013, accessed: 2016-11-01.

[40] Shumaker, B. P., and R. W. Sinnott. "Astronomical computing: 1. Computing under the open sky. 2. Virtues of the haversine.", Sky and telescope 68 (1984): 158-159.

Profit and Strategic Analysis for MNO-MVNO Partnership

Nesrine Ben Khalifa, Amal Benhamiche, Alain Simonian, Marc Bouillon
Orange Labs, France
Email: firstname.name@orange.com

***Abstract*—We consider a mobile market driven by two Mobile Network Operators (MNOs) and a new competitor Mobile Virtual Network Operator (MVNO). The MNOs can partner with the entrant MVNO by leasing network resources; however, the MVNO can also rely on other technologies such as free WiFi access points. Moreover, in addition to its connectivity offer, the MVNO can also draw indirect revenues from services due to its brand. In that framework including many access technologies and several revenue sources, a possible partner MNO will then have to decide which wholesale price to charge the MVNO for its resources. This multi-actor context, added to the need to consider both wholesale and retail markets, represents a new challenge for the underlying decision-making process. In this paper, the optimal price setting is formulated as a multi-level optimization problem which enables us to derive closed-form expressions for the optimal MNOs wholesale prices and the optimal MVNO retail price. The price attractivity of the MVNO is also evaluated in terms of its indirect revenues and the proportion of resources leased from possible partner MNOs. Finally, through a game-theoretical approach, we characterize the scenario where both MNOs partner with the MVNO as the unique Nash equilibrium under appropriate conditions.**

***Index Terms*—Network Economics, Strategic Partnership, Multi-Level Optimization, Non-Cooperative Game Theory**

I. Introduction

Mobile telecommunication markets are usually covered by a few number of operators because of high infrastructure and spectrum license costs. In addition, mobile operators are facing the challenges of upgrading their networks to 5G technology in order to cope with the increasing demand of users' traffic. This context, as well as the virtualization of wireless networks, may lower the barrier for the entrance of Mobile Virtual Network Operators (MVNO) to the market. While not possessing their own network infrastructure, MVNOs can lease capacities from Mobile Network Operators (MNOs) on the wholesale market to ensure wireless services to their customers.

A new generation of MVNOs has recently emerged, capable of leasing resources from different MNOs while also taking advantage of the available WiFi hotspots in order to propose a full connectivity offer to their customers. The latter can therefore afford to blindly use multiple network/technology services to establish the communications and use mobile Internet without any specific setting. Typically, such MVNOs can launch their activity thanks to specific inter-operator partnerships with MNOs for the cellular infrastructure utilization while the WiFi offloading part remains a unilateral decision. A recent example is offered by Google through its Project-Fi [1], launched in the U.S. in partnership with three leading MNOs, namely Sprint, T-Mobile and U.S. Cellular. This illustrates the case of a competitive MVNO able to absorb a significant part of the retail market (on mobile and data services) from the MNOs without being an expert in networking or even possessing the physical infrastructures and the ability to manage them.

On the other hand, an entrant MVNO can be already positioned at a higher place in the Value Chain and have income from its new customers through its current "non telco" activity. In fact, this MVNO may have a good reputation as an OTT ("Over the Top") providing content or other high level services; it can thus draw significant additional revenue, hereafter termed as *indirect* to differentiate it from the telco offer, from its new customers.

This new multi-technological and multi-activity context therefore raises new questions as to the interactions between MNOs and such a MVNO, namely the optimal price setting adopted by all actors and the consequences on the respective market shares and profits. The economic viability of a possible partnership between MNOs and the MVNO should, in particular, be understood on account of the existence of alternative technologies and other sources of revenues.

A. State-of-the-art

The ecosystem of MNO-MVNO relationships has been addressed in many studies. A detailed description of the MVNOs' classes in terms of their dependence to the host operator is presented in [2], where it is shown that a MVNO can be classified as either *light* or *full*. The authors describe, in particular, the possible business models of MVNOs and examine the impact of different parameters such as the MNO's market share on the outcome of cooperation between the host and virtual operators. In [3], the authors analyze the incentives for MNOs to form a strategic cooperation with MVNOs. They specifically examine the effects of the brand appeal of the MVNO and the wholesale discount offered by the MNO on the fulfillment of mutually beneficial partnerships.

Besides, the economic viability of MNO-MVNO relationship has been investigated in [4], [5] and references therein. First in [4], it is argued that the business of a MVNO may be profitable in a transitory phase when it partners with MNOs with a small market share; however, it is shown in [5] that, in the long term, the MVNO is better off when it preferably partners with a big MNO, say, the incumbent. The latter point

ISBN 978-3-903176-08-9

of view of a stabilized market and mature economic actors will be adopted in the sequel.

A multi-stage game for modeling the MNO-MVNO interaction is presented in [6] where the MNO investment, the MVNO's decision on the leasing from the MNO and the retail pricing are successively focused on at each stage. In [7], spectrum leasing and pricing are studied with game-theoretic models. A comprehensive study of the market share between MNO and MVNO based on the brand appeal of the MNO is provided in [8] where the authors also use game theory tools.

A recent study of Google-Fi like MVNOs is provided in [9] where the price setting between multiple service providers and the virtual operator is examined. In that paper, the user defection rate to the MVNO is assumed to be simply constant with no account of the impact of the MVNO price on its customer base; further, the authors only optimize the MVNO price for given wholesale MNO prices.

In contrast, we will here consider a more accurate economic model wherein (1) using the so-called price-demand elasticity, the users reply to the MVNO offer depends on the varying *price difference* between the MVNO offer and that of the other MNOs; (2) beside the search of an optimal retail price for the MVNO, we also determine the wholesale MNO prices by *maximizing their respective profit*.

B. Addressed issues and contributions

In this paper, we make a thorough economic analysis of strategic MNO-MVNO partnership. We consider the upcoming of a new MVNO, a potential competitor proposing low-cost services and threatening the MNO market share. Specifically, we address the following questions:

- When a MNO decides to conclude a partnership with the MVNO, what is the best price setting for the partner MNO in order to maximize its profit?
- When the wholesale prices are fixed by partner MNOs, what is the optimal price that should be charged by the MVNO to its customers ?
- What is the impact of the MVNO's indirect revenues on its optimal retail price ?
- What is the best decision for the MNOs facing the entry of the MVNO?

In this aim, the optimal price setting is addressed via a Stackelberg Game involving leaders and followers [5] (Section 2.3.6). In particular, we define and study two different decision-making models for the optimal wholesale price setting, namely a *fully sequential* model and a *partially sequential* model; we then determine the optimal retail price of the MVNO. Furthermore, we study the impact of the MVNO's indirect revenue on its optimal retail price in both decision-making models. Finally, we propose a game-theoretical approach to determine the Nash equilibrium under sufficient conditions on this indirect revenue and discuss its economic interpretation.

C. Paper structure

In Section II, we formally introduce the economic model describing the interactions between MNOs and the MVNO. In Section III, we formulate the price setting problem and study the wholesale and retail price optimization for all actors. A game-theoretical setting is discussed in Section IV. In Section V, we comment our general results on economic grounds. Finally, some concluding remarks are given in Section VI.

II. Economic Model and Assumptions

We consider a mobile operator market composed of two MNOs that share the whole customer base. The upcoming of a new MVNO proposing an attractive price impacts the repartition of customers who can defect from their original operators to the benefit of this MVNO. The MNOs must then identify the best decisions they are able to make, that is, to decide whether to partner with the MVNO. An operator that would decide to contract a partnership with the MVNO would obviously not be immune to lose customers but might, nevertheless, recover a part of the lost retail revenue via the wholesale income. Given this ecosystem, the economic variables of the MNOs and MVNO activity can be described as follows:

MNO's profit: the profit of a MNO is the sum of its revenues obtained from the retail market for end-users (and possibly from the Business-to-Business wholesale market for partner MNOs) from which are subtracted the corresponding costs. Denote by p_i (resp. w_i) the unit price of MNO i on the retail (resp. the wholesale) market; the retail revenues then depend linearly on the operator's customer base, while the wholesale revenues depend linearly on the amount of MVNO traffic accommodated by the MNO's network. Besides, we consider unit *network costs* c_i, *non-network costs* $\widetilde{c}_i$ (IT, commercial, etc.) and *fixed costs* $\overline{C}_i$; as a linear approximation, the total network and non-network costs are assumed to depend linearly on the customer base whereas fixed costs are independent of the amount of the operator's customer base;

MVNO's profit: the total revenue of the MVNO is the sum of *direct* revenues obtained from the retail market (with net unit price p_0) and of *indirect* revenues (with unit price r_0) obtained from e.g. advertising. As to network costs, we here consider that the MVNO enables its users' devices either to automatically connect to a free public WiFi Access Point (AP) or to a partner MNO's network, depending on the Quality of Service of each access mode. The proportion of MVNO traffic handled by free WiFi APs is denoted by $\gamma \in [0, 1[$; the only network costs of the MVNO are therefore those caused by the other proportion $1-\gamma$ of traffic dealt with MNOs through partnerships. In addition to these network costs, the MVNO finally incurs non-network and fixed costs denoted by $\widetilde{c}_0$ and $\overline{C}_0$, respectively.

For both MNOs, we assume that long-distance (backhaul and core) network costs are negligible compared to the access costs; the possible wholesale offer of any MNO will therefore mainly account for the transportation through its cellular access network of MVNO traffic.

Users Behavior: the users' reply to the MVNO's offer is assumed to depend only on the relative price of that offer. Specifically, the behavior of users is modeled according to

TABLE I
KEY TERMS AND SYMBOLS

Symbol	Definition (i = 1, 2)
p_i, c_i, $\widetilde{c}_i$	Unit retail price, network and non-network costs of MNO i
$\overline{C}_i$	Fixed costs of MNO i
Q_i	Customer base of MNO i before the MVNO's entry
Q	Total customer base
$\pi_i = Q_i/Q$	Market share of MNO i before the MVNO's entry
$Q_{i,0}$	Customer base of MNO i which defects to the MVNO
Q_0	Total customer base of the MVNO
r_0	Net unit indirect revenues of the MVNO
p_0, $\widetilde{c}_0$	Unit retail price, non-network costs of the MVNO
$\overline{C}_0$	Fixed costs of the MVNO
w_i	Unit wholesale price charged by MNO i to the MVNO
γ	Proportion of MVNO traffic handled by free WiFi APs

TABLE II
MVNO TRAFFIC SPLIT

Traffic Amount	Accommodation
γQ_0	WiFi APs
$(1-\gamma)\pi_1 Q_0$	MNO 1's Cellular BSs
$(1-\gamma)\pi_2 Q_0$	MNO 2's Cellular BSs

the price-demand elasticity [10] so that the part of MNO i customers which defects to the MVNO is expressed by

$$Q_{i,0} = \varepsilon\, Q_i \left(\frac{p_i - p_0}{p_i}\right) \tag{1}$$

where Q_i denotes the customer base of MNO i before the MVNO joins the market and $\varepsilon > 0$ is the price-demand elasticity coefficient (although generally depending on prices, ε is here assumed to be a constant on the basis of a small price variability range). Without loss of generality, we assume throughout the paper that $p_2 \leqslant p_1$ and $p_0 \leqslant p_2$; this ensures that both $Q_{1,0}$ and $Q_{2,0}$ are non-negative. The MVNO's total customer base is then

$$Q_0 = Q_{1,0} + Q_{2,0}. \tag{2}$$

MVNO traffic split: in the case when both MNOs partner with the MVNO, the MVNO traffic which is not supported by free WiFi APs is split between MNOs networks proportionally to their market share before the MVNO's upcoming. Consequently, the traffic transported for the MVNO on MNO i network equals $(1-\gamma)\pi_i Q_0$, where $\pi_i = Q_i/Q$ is the market share of MNO i before the MVNO's entry (with $\pi_1 + \pi_2 = 1$). This is motivated by the fact that the MVNO will partner with a MNO all the more that the latter has a large market share.

Note finally that a sample value of parameter γ is given by the ratio of the duration spent on WiFi access to the overall duration of a given communication session; this ratio can then be averaged over all successive sessions to provide the mean proportion γ; the latter is clearly related to the given geographic density of the WiFi APs.

Tables I and II sum up the notation used in the paper.

III. WHOLESALE AND RETAIL PRICE SETTING

In this section, we address the optimization of MNOs and MVNO profits in order to determine the optimal wholesale and retail prices. Specifically, we introduce several optimization problems where w_i, $i \in \{1, 2\}$, and p_0 are the decision variables. In the sequel, we denote by "Part" the strategy consisting in partnering with the MVNO and by "NonPart" the strategy consisting in not partnering with the MVNO. In a competitive environment where the MNOs do not collude with each other, two scenarios can be envisaged:

A. Only one operator, either MNO 1 or MNO 2, decides to contract with the MVNO. We denote this scenario by (Part-NonPart) or (NonPart-Part), respectively;

B. Both operators choose to contract with the MVNO. We denote this scenario by (Part-Part).

These two scenarios are successively analyzed below.

A. *Scenario (Part-NonPart)*

Consider first the case when only MNO i proposes a wholesale offer to the MVNO, while MNO $-i$ decides not to partner with the new entrant (by convention, i = 1 or 2 implies $-i$ = 2 or 1). In this case, the traffic γQ_0 generated by the MVNO's users will be delivered through free WiFi APs and the remaining $(1-\gamma)Q_0$ will be handled by the partner MNO's network. The MVNO's profit is therefore given by

$$\mathcal{R}_0(p_0, w_i) = (p_0 + r_0)Q_0 - w_i(1-\gamma)Q_0 - \widetilde{c}_0 Q_0 - \overline{C}_0. \tag{3}$$

The optimal MVNO's retail price p_0^* can then be determined by solving the following optimization problem:

$$\max_{0 \leqslant p_0 \leqslant p_2} \mathcal{R}_0(p_0, w_i). \tag{4}$$

Lemma 1. *In the (Part-NonPart) scenario, given the wholesale price w_i, the optimal MVNO retail price equals*

$$p_0^*(w_i) = \min(\widetilde{p}_0(w_i), p_2) \tag{5}$$

where

$$\widetilde{p}_0(w_i) = \frac{1-\gamma}{2} w_i + \frac{Q}{2\mathcal{S}} + \frac{\widetilde{c}_0 - r_0}{2}, \tag{6}$$

with $\mathcal{S} = Q_1/p_1 + Q_2/p_2$.

Proof. First observe that, in view of definitions (1) and (2), Q_0 involved in expression (3) is a linear function of variable p_0, thus making the profit $\mathcal{R}_0$ a quadratic function of p_0. Given the price w_i, the first order optimality condition $\partial \mathcal{R}_0(p_0, w_i)/\partial p_0 = 0$ for problem (4) then provides the critical point $\widetilde{p}_0(w_i)$ as given in (6). Besides, the second derivative $\partial^2 \mathcal{R}_0(p_0, w_i)/\partial p_0^2$ is equal to the negative constant $-2\varepsilon\mathcal{S}$; $\mathcal{R}_0(p_0, w_i)$ is therefore a strictly concave function of p_0 with a unique maximum at price $p_0^*(w_i)$ given by (5). □

Now, we consider the profit of MNO i given by

$$\begin{aligned}\mathcal{R}_i(p_0, w_i) = {} & p_i(Q_i - Q_{i,0}) + w_i(1-\gamma)Q_0 - \\ & c_i(Q_i - Q_{i,0} + (1-\gamma)Q_0) - \\ & \widetilde{c}_i(Q_i - Q_{i,0}) - \overline{C}_i,\end{aligned}$$

that is,

$$\mathcal{R}_i(p_0, w_i) = h_i(Q_i - Q_{i,0}) + (w_i - c_i)(1-\gamma)Q_0 - \overline{C}_i \quad (7)$$

where $h_i = p_i - c_i - \widetilde{c}_i \geqslant 0$. The partner MNO i seeks to maximize its profit $\mathcal{R}_i$. To this end, we replace p_0 involved in expression (7) via $Q_{i,0}$ and Q_0 by its optimal value $p_0^*(w_i)$ derived in Lemma 1; in fact, the MNO anticipates the best pricing strategy of the MVNO and thus sets its optimal wholesale price based on this anticipation. Define then

$$\mathcal{R}_i^*(w_i) = \mathcal{R}_i(p_0^*(w_i), w_i) \quad (8)$$

which is obtained through (7) with $Q_{i,0} = \varepsilon Q_i(p_i - p_0^*(w_i))/p_i$ and Q_0 given by (2). The optimization problem for MNO i can then be expressed by

$$\max_{w_i \geqslant 0} \mathcal{R}_i^*(w_i). \quad (9)$$

Proposition 1. *In the (Part-NonPart) scenario, the optimal MNO's wholesale price equals*

$$\widehat{w}_i = \min(\overline{w}_i, \widetilde{w}_i) \quad (10)$$

where we set

$$\overline{w}_i = \frac{1}{1-\gamma}\left(2p_2 - \frac{Q}{\mathcal{S}} + r_0 - \widetilde{c}_0\right) \quad (11)$$

and

$$\widetilde{w}_i = \frac{c_i}{2} + \frac{1}{1-\gamma}\left(\frac{h_i Q_i}{2p_i\mathcal{S}} + \frac{Q}{2\mathcal{S}} + \frac{r_0 - \widetilde{c}_0}{2}\right). \quad (12)$$

The optimal MVNO's retail price is then determined by

$$\widehat{p}_0(\widehat{w}_i) = \min(\widetilde{p}_0(\widehat{w}_i), p_2). \quad (13)$$

Defining the constant

$$\overline{r}_{i,0} = \frac{h_i Q_i}{p_i\mathcal{S}} + c_i(1-\gamma) + \frac{3Q}{\mathcal{S}} + \widetilde{c}_0 - 4p_2, \quad (14)$$

we then have $\overline{w}_i \leqslant \widetilde{w}_i \iff p_0^*(\widehat{w}_i) = p_2 \iff r_0 \leqslant \overline{r}_{i,0}$.

Proof. We consider the two cases (a) $\widetilde{p}_0(w_i) > p_2$ and (b) $\widetilde{p}_0(w_i) \leqslant p_2$. First consider case (a). This corresponds to values of w_i such that $w_i > \overline{w}_i$ where $\overline{w}_i$ is given by (11). Relation (5) then yields $p_0^*(w_i) = p_2$ and by (8), we easily show that $\mathcal{R}_i^*(w_i) = \mathcal{R}_i(p_2, w_i)$ is a linear and increasing function of w_i. The optimal value of $\mathcal{R}_i^*$ is thus obtained at $w_i = +\infty$; but this unbounded price giving the value $-\infty$ for $\mathcal{R}_0$, case (a) is thus eventually excluded.

Now consider case (b). This corresponds to values of w_i such that $w_i \leqslant \overline{w}_i$. Relation (5) now yields $p_0^*(w_i) = \widetilde{p}_0(w_i)$; using (8) again and writing now

$$Q_{i,0} = \varepsilon\frac{Q_i}{p_i}(p_i - \widetilde{p}_0(w_i)), \quad Q_0 = Q_{1,0} + Q_{2,0}$$

as functions of w_i, $\mathcal{R}_i^*(w_i) = \mathcal{R}_i(\widetilde{p}_0(w_i), w_i)$ can then be easily expressed as a quadratic function of w_i. The first order optimality condition $\partial\mathcal{R}_i^*(w_i)/\partial w_i = 0$ for problem (9) yields the critical point $\widetilde{w}_i$ as given in (12). Besides, the second derivative $\partial^2\mathcal{R}_i^*(w_i)/\partial w_i^2$ is equal to the negative constant $-\varepsilon(1-\gamma)^2\mathcal{S}/2$. Therefore, $\mathcal{R}_i^*(w_i)$ is a strictly concave function of w_i with a unique maximum at $\widehat{w}_i$ given by (10).

The optimal MVNO's retail price is obtained by replacing w_i in (5) by $\widehat{w}_i$ given in (10), hence (13). Finally, elementary algebra reduces condition $\overline{w}_i \leqslant \widetilde{w}_i$ to $r_0 \leqslant \overline{r}_{i,0}$, with $\overline{r}_{i,0}$ given by (14). □

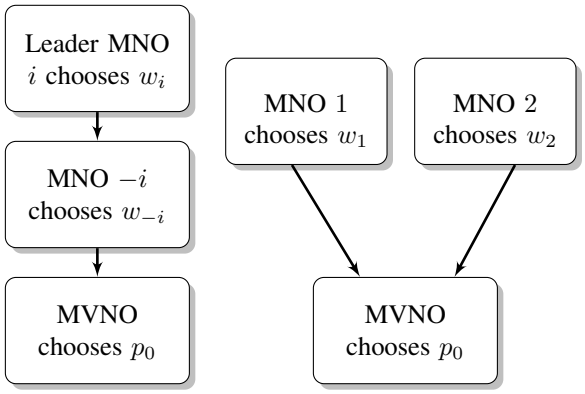


Fig. 1. Hierarchical decision models. *Left,* Fully sequential model. *Right,* Partially sequential model.

All previous results symmetrically hold for the (NonPart-Part) scenario.

B. Scenario (Part-Part)

We now turn to the situation where both MNOs partner with the MVNO. Two models can be proposed depending on the order in which decisions are taken, namely:

- A *Fully Sequential* (FS) model where a leader MNO (say, the one with the highest market share) first chooses its wholesale price; then the second MNO, the follower, determines its wholesale price accordingly; finally, the MVNO chooses its retail price. This situation is illustrated in Fig. 1-*Left*.
- A *Partially Sequential* (PS) model where first the two MNOs (say, with comparable weights) choose their respective wholesale price without coordination; then, the MVNO chooses its retail price. This situation is illustrated in Fig. 1-*Right*.

In both models, we consider that the MNOs move before the MVNO because they own the resources to lease to the latter. They forecast the MVNO's reply to their pricing strategies and choose the wholesale prices that maximize their profits, given the anticipated MVNO's optimal pricing strategy. The MVNO eventually chooses its retail price, given the wholesale prices fixed by the MNOs.

The MVNO's profit is now given by

$$\begin{aligned}\mathcal{R}_0(p_0, w_1, w_2) \quad &= \quad (p_0 + r_0)Q_0 - w_1(1-\gamma)\pi_1 Q_0 - \\ &\qquad w_2(1-\gamma)\pi_2 Q_0 - \widetilde{c}_0 Q_0 - \overline{C}_0 \quad (15)\end{aligned}$$

and the optimization problem of the MVNO is formulated by

$$\max_{0 \leqslant p_0 \leqslant p_2} \mathcal{R}_0(p_0, w_1, w_2). \quad (16)$$

Lemma 2. *In the (Part-Part) scenario, given the wholesale prices w_1 and w_2, the optimal retail price of the MVNO equals*

$$p_0^*(w_1, w_2) = \min(\widetilde{p}_0(w_1, w_2), p_2) \tag{17}$$

where

$$\widetilde{p}_0(w_1, w_2) = \frac{(1-\gamma)}{2}(\pi_1 w_1 + \pi_2 w_2) + \frac{Q}{2\mathcal{S}} + \frac{\widetilde{c}_0 - r_0}{2}. \tag{18}$$

Proof. Similar to that of Lemma 1. □

In this section, the notations for profit $\mathcal{R}_i$ should not be confused with that of Section III-A. Now, consider the profit of MNO $i \in \{1, 2\}$ given by

$$\begin{aligned}\mathcal{R}_i(p_0, w_i) \quad = \quad & p_i(Q_i - Q_{i,0}) + w_i(1-\gamma)\pi_i Q_0 - \\ & c_i(Q_i - Q_{i,0} + (1-\gamma)\pi_i Q_0) - \\ & \widetilde{c}_i(Q_i - Q_{i,0}) - \overline{C}_i,\end{aligned}$$

that is,

$$\begin{aligned}\mathcal{R}_i(p_0, w_i) = h_i(Q_i - Q_{i,0}) + \\ (w_i - c_i)(1-\gamma)\pi_i Q_0 - \overline{C}_i\end{aligned} \tag{19}$$

where we set $h_i = p_i - c_i - \widetilde{c}_i$. Both MNO partners 1 and 2 seek to maximize their own profit $\mathcal{R}_1$ and $\mathcal{R}_2$. In order to solve this optimization problem for either model (FS) or (PS), we replace p_0 by its optimal value $p_0^*(w_1, w_2)$ derived above in Lemma 2. Define then

$$\mathcal{R}_i^*(w_1, w_2) = \mathcal{R}_i(p_0^*(w_1, w_2), w_i) \tag{20}$$

as obtained from (19) with $Q_{i,0} = \varepsilon Q_i(p_i - p_0^*(w_1, w_2))/p_i$ and Q_0 given by (2).

We now successively address the maximization of MNOs profits for the (FS) and (PS) models.

*1) **Fully Sequential Model**:* Assume that MNO $i \in \{1, 2\}$ is the leader and MNO $-i$ is the follower (by convention, $-i = 2$ if $i = 1$, and $-i = 1$ if $i = 2$). Given w_i, the follower thus decides on the wholesale price w_{-i} to charge the MVNO. The optimization problem for both MNOs can then be expressed by the following bilevel formulation

$$\begin{cases} \max\limits_{w_i \geqslant 0} \mathcal{R}_i^*(w_1, w_2)_{|w_{-i} = w_{-i}^*}, \\ \text{subject to} \qquad w_{-i}^* = \operatorname{argmax}\limits_{w_{-i} \geqslant 0} \mathcal{R}_{-i}^*(w_1, w_2) \end{cases} \tag{21}$$

where the notation $|w_{-i} = w_{-i}^*$ means that function $\mathcal{R}_i^*$ is evaluated for the variable w_{-i} equal to w_{-i}^*; note that w_{-i}^* is a function of w_i. The symmetrical case when MNO $-i$ is the leader is similarly defined. In order to solve problem (21), we introduce the following definitions.

Definition 1. *We denote by Δ the closed triangular region defined by $\Delta = \{(w_1, w_2) \in \mathbb{R}^+ \times \mathbb{R}^+ : \widetilde{p}_0(w_1, w_2) \leqslant p_2\}$ where $\widetilde{p}_0(w_1, w_2)$ is given by* (18) *(see Fig. 2).*

Further denote by δ the boundary segment of Δ defined by $\delta = \{(w_1, w_2) \in \Delta : \widetilde{p}_0(w_1, w_2) = p_2\}$.

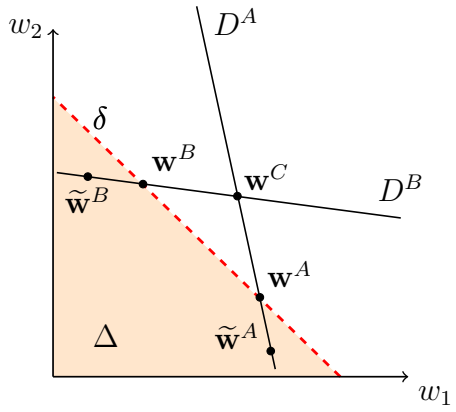


Fig. 2. The region Δ, its boundary δ, and the points $\mathbf{w}^A$, $\mathbf{w}^B$, $\widetilde{\mathbf{w}}^A$, $\widetilde{\mathbf{w}}^B$ and $\mathbf{w}^C$ (lines D^A and D^B are invoked in Appendix VII-A).

Definition 2. *Let $\mathbf{w}^A = (w_1^A, w_2^A)$ and $\mathbf{w}^B = (w_1^B, w_2^B)$ denote the pair of prices given by*

$$\begin{cases} w_1^A = \dfrac{\dfrac{h_1 Q_1}{p_1} + c_1(1-\gamma)\pi_1\mathcal{S} + 2Q - 2\mathcal{S}p_2}{(1-\gamma)\pi_1\mathcal{S}}, \\ w_2^A = \dfrac{-\dfrac{h_1 Q_1}{p_1} - c_1(1-\gamma)\pi_1\mathcal{S} - 4Q + 4p_2\mathcal{S} + \mathcal{T}}{(1-\gamma)\pi_2\mathcal{S}} \end{cases}$$

and

$$\begin{cases} w_1^B = \dfrac{-\dfrac{h_2 Q_2}{p_2} - c_2(1-\gamma)\pi_2\mathcal{S} - 4Q + 4p_2\mathcal{S} + \mathcal{T}}{(1-\gamma)\pi_1\mathcal{S}}, \\ w_2^B = \dfrac{\dfrac{h_2 Q_2}{p_2} + c_2(1-\gamma)\pi_2\mathcal{S} + 2Q - 2\mathcal{S}p_2}{(1-\gamma)\pi_2\mathcal{S}}, \end{cases}$$

respectively, with $\mathcal{T} = Q + (r_0 - \widetilde{c}_0)\mathcal{S}$ for short.

Define also the function Ω_{-i}, $i \in \{1, 2\}$, by

$$\begin{aligned}\Omega_{-i}(w_i) \quad = \quad & \frac{c_{-i}}{2} + \frac{h_{-i}Q_{-i}}{2p_{-i}(1-\gamma)\pi_{-i}\mathcal{S}} + \frac{Q}{2(1-\gamma)\pi_{-i}\mathcal{S}} - \\ & \frac{\widetilde{c}_0 - r_0}{2(1-\gamma)\pi_{-i}} - \frac{\pi_i}{2\pi_{-i}} w_i\end{aligned} \tag{22}$$

and the points $\widetilde{\mathbf{w}}^B = (\widetilde{w}_1, \Omega_2(\widetilde{w}_1))$, $\widetilde{\mathbf{w}}^A = (\Omega_1(\widetilde{w}_2), \widetilde{w}_2)$ where

$$\widetilde{w}_i = \frac{\dfrac{h_i Q_i}{p_i} - \dfrac{h_{-i}Q_{-i}}{p_{-i}} + \mathcal{S}(1-\gamma)(c_i\pi_i - c_{-i}\pi_{-i}) + \mathcal{T}}{2(1-\gamma)\pi_i\mathcal{S}}. \tag{23}$$

Note that $\mathbf{w}^A \in \delta$ and $\mathbf{w}^B \in \delta$; a geometric interpretation of the pairs $\mathbf{w}^A$, $\mathbf{w}^B$ and $\widetilde{\mathbf{w}}^A$, $\widetilde{\mathbf{w}}^B$ is given in Appendix VII-A. We can now state the following.

Proposition 2. *Define the constant $\overline{r}_0$ by*

$$\overline{r}_0 = \frac{h_1 Q_1}{p_1\mathcal{S}} + \frac{h_2 Q_2}{p_2\mathcal{S}} + (1-\gamma)(\pi_1 c_1 + \pi_2 c_2) + \frac{7Q}{\mathcal{S}} + \widetilde{c}_0 - 8p_2.$$

In the (Part-Part) scenario with FS model,

- *if $r_0 \leqslant \overline{r}_0$ and MNO 1 (resp. MNO 2) is the leader, then the optimal wholesale price vector (w_1^*, w_2^*) is given by $\boldsymbol{w}^B \in \delta$ (resp. $\boldsymbol{w}^A \in \delta$) introduced above. In either case, the optimal MVNO's retail price is then*

$$p_0^*(w_1^*, w_2^*) = p_2;$$

- *if $r_0 > \overline{r}_0$ and MNO 1 (resp. MNO 2) is the leader, then the optimal wholesale price vector (w_1^*, w_2^*) is given by $\widetilde{\mathbf{w}}^B \in \Delta \setminus \delta$ (resp. $\widetilde{\mathbf{w}}^A \in \Delta \setminus \delta$) introduced above. In either case, the optimal MVNO's retail price is then*

$$p_0^*(w_1^*, w_2^*) = \widetilde{p}_0(w_1^*, w_2^*) < p_2.$$

We defer the detailed proof to Appendix VII-A.

*2) **Partially Sequential Model**:* Now assume that MNOs simultaneously choose their optimal wholesale prices. Given (20), the joint optimization problem for both MNOs is thus expressed by

$$\max_{w_1 \geqslant 0} \mathcal{R}_1^*(w_1, w_2) \tag{24}$$

and

$$\max_{w_2 \geqslant 0} \mathcal{R}_2^*(w_1, w_2). \tag{25}$$

Definition 3. *Let $\mathbf{w}^C = (w_1^C, w_2^C)$ denote the pair of prices given by*

$$\begin{cases} w_1^C = \dfrac{\dfrac{2h_1\,Q_1}{p_1} - \dfrac{h_2\,Q_2}{p_2} + (1-\gamma)(2c_1\pi_1 - c_2\pi_2)\mathcal{S} + \mathcal{T}}{3(1-\gamma)\pi_1\mathcal{S}}, \\ w_2^C = \dfrac{\dfrac{2h_2\,Q_2}{p_2} - \dfrac{h_1\,Q_1}{p_1} + (1-\gamma)(2c_2\pi_2 - c_1\pi_1)\mathcal{S} + \mathcal{T}}{3(1-\gamma)\pi_2\mathcal{S}} \end{cases}$$

with $\mathcal{T} = Q + (r_0 - \widetilde{c}_0)\mathcal{S}$.

The pair $\mathbf{w}^C$ is given a geometric interpretation in Appendix VII-B. This enables us to state the following.

Proposition 3. *Define the constant $r_0^\flat$ by*

$$r_0^\flat = \frac{h_1 Q_1}{p_1\mathcal{S}} + \frac{h_2 Q_2}{p_2\mathcal{S}} + (1-\gamma)(\pi_1 c_1 + \pi_2 c_2) + \frac{5\,Q}{\mathcal{S}} + \widetilde{c}_0 - 6p_2.$$

In the (Part-Part) scenario with PS model,

- *if $r_0 > r_0^\flat$, the optimal wholesale price vector (w_1^*, w_2^*) is given by $\mathbf{w}^C \in \Delta \setminus \delta$. The optimal MVNO's retail price is then*

$$p_0^*(w_1^C, w_2^C) = \widetilde{p}_0(w_1^C, w_2^C) < p_2$$

  as defined by (18);
- *if $r_0 = r_0^\flat$, the optimal wholesale price vector (w_1^*, w_2^*) is given by $\mathbf{w}^C \in \delta$. The optimal MVNO's retail price is then $p_0^* = p_2$;*
- *if $r_0 < r_0^\flat$, the problem* (24)-(25) *admits no solution.*

The proof of Proposition 3 is detailed in Appendix VII-B.

IV. Game-Theoretic Model

In this section, we propose a non-cooperative game-theoretical model to formalize the competition between MNO 1 and MNO 2.

Definition 4. *Introduce the two-player game where*

- *the players are MNO 1 and MNO 2,*
- *the strategies are either to contract with the new entrant, "Part" strategy, or not to contract, "NonPart" strategy,*
- *the payoffs are given by the matrix*

$$\begin{array}{c} \\ Part \\ NonPart \end{array} \begin{array}{c} \begin{array}{cc} Part & \quad NonPart \end{array} \\ \begin{pmatrix} (\mathcal{R}_1^*(w_1^*, w_2^*), \mathcal{R}_2^*(w_1^*, w_2^*)) & (\mathcal{R}_1^*(\widehat{w}_1), \mathcal{R}_2(\widehat{w}_1)) \\ (\mathcal{R}_1(\widehat{w}_2), \mathcal{R}_2^*(\widehat{w}_2)) & (\mathcal{R}_1^0, \mathcal{R}_2^0) \end{pmatrix} \end{array}.$$

In this matrix, the pair $(\mathcal{R}_1^0, \mathcal{R}_2^0)$ corresponds to the scenario when no MNO partners with the MVNO, thus forbidding its entrance into the market.

We recall that a Nash Equilibrium (NE) is a strategy profile such that no player has an incentive to unilaterally deviate from this profile [11]. Besides, recall from definition (1) that $Q_{i,0}$ denotes the customer part which defects from MNO i to the new entrant in the scenario (Part, Part); furthermore, let $\widehat{Q}_{i,0}$ denote the customer part which defects from MNO i to the MVNO in the scenario (NonPart, Part), that is, when only MNO $-i$ contracts a partnership with the new entrant.

Lemma 3. *In the FS model with $r_0 \leqslant \overline{r}_0$, a MNO loses more customers when it is non-partner than when it is a partner of the MVNO, that is, $\widehat{Q}_{i,0} \geqslant Q_{i,0}$.*

Proof. From Proposition 2 with $r_0 \leqslant \overline{r}_0$, we have $p_0^*(w_1^*, w_2^*) = p_2$ while Proposition 1 entails $\widehat{p}_0(\widehat{w}_{-i}) \leqslant p_2$. Using definition (1), we then deduce that the difference $Q_{i,0} - \widehat{Q}_{i,0} = \frac{\varepsilon Q_i}{p_i}\left(\widehat{p}_0(\widehat{w}_{-i}) - p_0^*(w_1^*, w_2^*)\right)$ is non-positive, as claimed. □

As a consequence, we can formulate the subsequent result on the existence of a Nash equilibrium.

Proposition 4. *In the FS model with $r_0 \leqslant \overline{r}_0$ and for wholesale prices higher than network costs,*

(a) the scenario (Part, Part) is a NE;

(b) if $r_0 \leqslant \min(\overline{r}_0, \overline{r}_{2,0})$, (Part, Part) is the unique NE.

Proof. (a) Scenario (Part, Part) is a NE. Suppose that both operators partner with the MVNO, that is, the scenario is (Part, Part); consider MNO i and let $\mathcal{R}_i^*(w_1^*, w_2^*)$ denote its profit for this scenario. Assume now that MNO i unilaterally switches from strategy "Part" to strategy "NonPart"; we denote by $\mathcal{R}_i(\widehat{w}_{-i})$ the profit of MNO i when it applies strategy "NonPart" while MNO $-i$ keeps strategy "Part". We then have

$$\mathcal{R}_i^*(w_1^*, w_2^*) = h_i(Q_i - Q_{i,0}) + (w_i^* - c_i)(1-\gamma)\pi_i Q_0 - \overline{C}_i$$

and $\mathcal{R}_i(\widehat{w}_{-i}) = h_i(Q_i - \widehat{Q}_{i,0}) - \overline{C}_i$, so that

$$\mathcal{R}_i^*(w_1^*, w_2^*) - \mathcal{R}_i(\widehat{w}_{-i}) = h_i(\widehat{Q}_{i,0} - Q_{i,0}) + (w_i^* - c_i)\times (1-\gamma)\pi_i Q_0. \tag{26}$$

As $\widehat{Q}_{i,0} \geqslant Q_{i,0}$ by Lemma 3, the profit difference $\mathcal{R}_i^*(w_1^*, w_2^*) - \mathcal{R}_i(\widehat{w}_{-i})$ in (26) is then non-negative after assumptions $h_i \geqslant 0$ and $w_i^* - c_i \geqslant 0$; MNO i then has no incentive to unilaterally deviate from strategy "Part" since this yields a profit decrease. (Part, Part) is thus a NE.

(b) (Part, Part) is the unique NE. Assume both MNOs use strategy "Non Part". Now suppose MNO 2 unilaterally switches to "Part"; we first have $\mathcal{R}_2^0 = h_2 Q_2 - \overline{C}_2$ while $\mathcal{R}_2^*(\widehat{w}_2) = h_2(Q_2 - \widehat{Q}_{2,0}) + (\widehat{w}_2 - c_2)(1-\gamma)Q_0 - \overline{C}_2$ after (7), hence

$$\mathcal{R}_2^*(\widehat{w}_2) - \mathcal{R}_2^0 = -h_2\widehat{Q}_{2,0} + (\widehat{w}_2 - c_2)(1-\gamma)Q_0. \quad (27)$$

From Proposition 1, we have $\widehat{Q}_{2,0} = 0$ if $r_0 \leqslant \overline{r}_{2,0}$; the difference $\mathcal{R}_2^*(\widehat{w}_2) - \mathcal{R}_2^0$ in (27) is then non-negative after assumption $\widehat{w}_2 - c_2 \geqslant 0$ and (Non Part, Non Part) is not a NE. We conclude that if $r_0 \leqslant \overline{r}_0$ and $r_0 \leqslant \overline{r}_{2,0}$, (Part,Part) is the only NE. □

Note that a preliminary study has given us hints for (Part,Part) to be still a NE for $r_0 > \overline{r}_0$. For the PS model, a similar analysis should also be addressed for $r_0 \leqslant r_0^\flat$.

V. Economic Discussion

The results obtained in the previous sections allow us to provide the following comments.

Scenario (Part-Part): When both MNOs partner with the MVNO, Propositions 2 and 3 entail that, when $\gamma \to 1$, all optimal wholesale prices are of order $1/(1-\gamma)$; this increasing rate again confirms the effect of economy of scale. Now, regarding the MVNO's optimal retail price, we distinguish two cases depending on the order in which decisions are taken by MNOs. First, we have shown (Proposition 2) for the FS model that the entrant MVNO charges its users the same price p_2 as the lowest MNO if $r_0 \leqslant \overline{r}_0$, thus attracting users only from the MNO with the highest retail price; otherwise, it attracts users from both MNOs. Second, we have shown (Proposition 3) for the PS model that the entrant MVNO cannot set an optimal retail price strictly lower than both MNOs' prices unless it has sufficiently high indirect revenues, that is, $r_0 > r_0^\flat$. Otherwise, if $r_0 < r_0^\flat$, there is no wholesale prices that jointly optimize both MNOs profits.

For each class of actors, we can conclude the following:

For the MVNO: for all scenarios, the MVNO retail price p_0^* decreases with r_0; the MVNO can thus set a retail price strictly lower than that of the MNOs if it has high enough indirect revenues. Besides, the threshold value $\overline{r}_0$ (resp. $r_0^\flat$) in model FS (resp. model PS) is a decreasing function of the proportion γ, so that the MVNO has an optimal retail price strictly lower than that of both MNOs for large enough γ. The technological independence of the MVNO from its partner MNOs due to free WiFi access thus translates into an economic advantage on the retail market (but this does not obviously account for better QoS and security levels offered to its customers if a larger part of MVNO traffic were transferred through optimized cellular networks);

For the MNOs: for the FS model and under sufficient conditions on the indirect revenue of the MVNO, the scenario (Part, Part) defines the unique NE. This means that the MNOs have then an incentive to partner with the new entrant: in fact, the MNOs would in the first place prefer that the MVNO does not enter the market in order to keep their customer base, but each MNO fears that its competitor hosts the MVNO, in which case it would incur losses both on the retail and wholesale markets. As a consequence, both MNOs will eventually decide to partner with the MVNO. In addition, a non-partner MNO would incur higher retail losses if it were non-partner than when it is a partner of the MVNO. Cooperating with the MVNO therefore enables each MNO to compensate for a part of its retail revenue losses.

VI. Conclusion

In this paper, our contribution is twofold. First, we address the *price setting optimization problems* for both MNOs and the entrant MVNO in the framework of two distinct scenarios. In this aim, we propose several mathematical programming formulations for the underlying problems, each one corresponding to a specific decision-making scheme; we also discuss the economic interpretation of the optimal solution in each case. Secondly, based on the optimal price setting step, we provide a *game-theoretical analysis* of the MNOs competition and show that (Part, Part) is the unique Nash Equilibrium, and thus the most profitable scenario for both MNOs, provided that appropriate conditions on the MVNO's indirect revenue are fulfilled.

The particular case of only two competing MNOs has provided us with interesting results, and a natural extension to this work would be to generalize the results obtained for two MNOs to an arbitrary number ($n \geqslant 3$) of MNOs. Indeed, the two-dimensional optimization problems that we have here addressed may exhibit other features in the n-dimensional case (e.g. several possible optimal points). Furthermore, the associated n-player games could be amenable to cooperation schemes among players which could be interestingly studied. On the other hand, other demand models differing from that considered in this paper (customer defection due to the price/demand elasticity) could also be envisaged; alternative models based on interactions between users or on the MVNO brand appeal could capture other preference sources of users towards each actor.

Acknowledgment

The authors thank V. Chandrakumar, L. Le Beller and M. Touati at Orange Labs for fruitful discussions, together with the anonymous referees for their valuable comments.

References

[1] "Google's project Fi website," https://fi.google.com/about/.

[2] M. Balon and B. Liau, "Mobile virtual network operator," in *Proc. of the 15th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, Rome, Oct 2012, pp. 1–6.

[3] A. Banerjee and C. M. Dippon, "Voluntary relationships among mobile network operators and mobile virtual network operators: An economic explanation," *Information Economics and Policy*, vol. 21, no. 1, pp. 72–84, 2009.
[4] L. Zheng, C. Joe-Wong, J. Chen, C. G. Brinton, C. W. Tan, and M. Chiang, "Economic viability of a virtual ISP," in *Proc. of the IEEE International Conference on Computer Communications (Infocom)*, Atlanta, USA, May 2017, pp. 1–9.
[5] P. Maillé and B. Tuffin, *Telecommunication Network Economics From Theory to Applications*. Cambridge, 2014.
[6] M. H. Lotfi and S. Sarkar, "The economics of competition and cooperation between MNOs and MVNOs," in *Annual Conference on Information Science and Systems (CISS)*, Baltimore, March 2017.
[7] L. Guijarro, V. Pla, B. Tuffin, P. Maille, and J. R. Vidal, "Competition and bargaining in wireless networks with spectrum leasing," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, Houston, USA, Dec 2011, pp. 1–6.
[8] M. Debbah, L. Echabbi, and C. Hamlaoui, "Market share analysis between MNO and MVNO under brand appeal based segmentation," in *6th International Conference on Network Games, Control and Optimization (NetGCooP)*, Avignon, France, Nov. 2012, pp. 9–16.
[9] L. Zheng, J. Chen, C. Joe-Wong, C. W. Tan, and M. Chiang, "An economic analysis of wireless network infrastructure sharing," in *Proc. of the 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, Paris, May 2017, pp. 1–8.
[10] K. E. Case, R. C. Fair, and S. M. Oster, *Principles of Economics*. Pearson Prentice Hall, 2012.
[11] J. Nash, "Non-cooperative games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, Sep. 1951.
[12] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms, Third edition*. New Jersey: Wiley, 2006.

VII. Appendix

A. *Proof of Proposition 2.*

Recall that MNO $i \in \{1,2\}$ (resp. MNO $-i$) is assumed to be the leader (resp. the follower). We successively consider the two cases (a) $(w_1,w_2) \notin \Delta$ and (b) $(w_1,w_2) \in \Delta$.

First consider case (a). In view of (17), this corresponds to values of (w_1,w_2) such that $p_0^*(w_1,w_2) = p_2$ and by (20), we easily show that $\mathcal{R}_{-i}^*(w_1,w_2) = \mathcal{R}_{-i}(p_2,w_{-i})$ is a linear and increasing function of w_{-i}. The optimal value of $\mathcal{R}_{-i}^*$ is thus obtained at $w_{-i} = +\infty$; as this unbounded price gives the value $-\infty$ for $\mathcal{R}_0$, case (a) can thus be excluded.

Now consider case (b). By (17), this corresponds to pairs (w_1,w_2) such that $p_0^*(w_1,w_2) = \widetilde{p}_0(w_1,w_2)$. To solve the optimization problem for follower MNO $-i$ in (21) with given w_i, we introduce the associated Lagrange function given by

$$\mathcal{L}_{-i}(w_1,w_2,\lambda_{-i}) = \mathcal{R}_{-i}^*(w_1,w_2) - \lambda_{-i}\left(\widetilde{p}_0(w_1,w_2) - p_2\right)$$

where λ_{-i} is the Lagrange multiplier associated to the constraint $\widetilde{p}_0(w_1,w_2) \leqslant p_2$; from definition (20) and the expression (18) of $\widetilde{p}_0(w_1,w_2)$, $\mathcal{R}_{-i}^*(w_1,w_2)$ is easily expressed as a quadratic function of variable w_{-i}. The system of Karush-Kuhn-Tucker (KKT) ([12], Chap.4, Sec. 4.2.13) conditions for the Lagrangian $\mathcal{L}_{-i}$ above can be written as

$$\begin{cases} \dfrac{\partial \mathcal{L}_{-i}}{\partial w_{-i}}(w_1,w_2,\lambda_{-i}) = 0, \\ \\ \lambda_{-i} \geqslant 0, \qquad \lambda_{-i}\big(\widetilde{p}_0(w_1,w_2) - p_2\big) = 0, \\ \\ \widetilde{p}_0(w_1,w_2) \leqslant p_2. \end{cases} \tag{28}$$

Two cases can intervene for the multiplier λ_{-i}:

(I) if $\lambda_{-i} = 0$, the first KKT condition in system (28) reads

$$\frac{\partial \mathcal{R}_{-i}^*}{\partial w_{-i}}(w_1,w_2) = 0; \tag{29}$$

function $\mathcal{R}_{-i}^*$ being quadratic, equation (29) is linear in both variables w_1, w_2 and thus defines geometrically a line D (displayed in Fig. 2 as line D^B if $-i = 2$ or line D^A if $-i = 1$). Solving (29) for w_{-i} then yields the unique maximum at point $w_{-i}^* = \Omega_{-i}(w_i)$ with function Ω_{-i} defined as in (22).

Now, consider the profit maximization in problem (21) for the leader MNO i. In order to solve it, we replace w_{-i} in the profit $\mathcal{R}_i^*(w_1,w_2)$ by the maximum $w_{-i}^* = \Omega_{-i}(w_i)$ derived above. Given (20), we thus define

$$\mathcal{R}_i^{**}(w_i) = \mathcal{R}_i^*(w_1,w_2)_{|w_{-i}=\Omega_{-i}(w_i)}.$$

Recall that we consider case (b) for which $(w_1,w_2) \in \Delta$ and $p_0^*(w_1,w_2) = \widetilde{p}_0(w_1,w_2)$, so that the optimization problem for leader MNO i eventually reads

$$\begin{cases} \max\limits_{w_i \geqslant 0} \mathcal{R}_i^{**}(w_i), \\ \\ \text{subject to} \quad \widetilde{p}_0(w_1,w_2)_{|w_{-i}=\Omega_{-i}(w_i)} \leqslant p_2. \end{cases} \tag{30}$$

First, the constraint $\widetilde{p}_0(w_1,w_2)_{|w_{-i}=\Omega_{-i}(w_i)} \leqslant p_2$ in (30) is easily translated into $w_i \leqslant \overline{w}_i$ where we set

$$\overline{w}_i = \frac{-\dfrac{h_{-i}Q_{-i}}{p_{-i}} - c_{-i}(1-\gamma)\pi_{-i}\mathcal{S} - 4Q + 4p_2\mathcal{S} + \mathcal{T}}{(1-\gamma)\pi_i\mathcal{S}}$$

with $\mathcal{T} = Q + (r_0 - \tilde{c}_0)\mathcal{S}$. Second, the 1st order condition $\partial \mathcal{R}_i^{**}(w_i)/\partial w_i = 0$ for problem (30) yields the critical point $\widetilde{w}_i$, given as in (23). The second derivative $\partial^2 \mathcal{R}_i^{**}(w_i)/\partial w_i^2$ being equal to the negative constant $-\varepsilon(1-\gamma)^2\pi_i^2\mathcal{S}/2$, $\mathcal{R}_i^{**}(w_i)$ is therefore a strictly concave function of w_i and has a unique maximum on $\mathbb{R}^+$ at $\widetilde{w}_i$. It thus follows from the latter discussion that

$$w_i^* = \min(\widetilde{w}_i, \overline{w}_i)$$

is the unique solution to problem (30). Now, we easily verify that $\widetilde{w}_i \geqslant \overline{w}_i \iff r_0 \leqslant \overline{r}_0$ where $\overline{r}_0$ is expressed in Proposition 2. We thus conclude that if $r_0 \leqslant \overline{r}_0$, the optimal solution is the intersection point $\mathbf{w}^* = (\overline{w}_1, \Omega_2(\overline{w}_1)) = \mathbf{w}^B \in D^B \cap \delta$ when $i = 1$, or $\mathbf{w}^* = (\Omega_1(\overline{w}_2), \overline{w}_2) = \mathbf{w}^A \in D^A \cap \delta$ when $i = 2$; otherwise, if $r_0 > \overline{r}_0$, the optimal solution is the intersection point $\mathbf{w}^* = (\widetilde{w}_1, \Omega_2(\widetilde{w}_1)) = \widetilde{\mathbf{w}}^B \in D^B \cap \Delta$ when $i = 1$, or $\mathbf{w}^* = (\Omega_1(\widetilde{w}_2), \widetilde{w}_2) = \widetilde{\mathbf{w}}^A \in D^A \cap \Delta$ when $i = 2$;.

(II) otherwise, if $\lambda_{-i} > 0$, the first KKT condition $\partial \mathcal{L}_{-i}(w_1,w_2,\lambda_{-i})/\partial w_{-i} = 0$ in system (28) yields

$$w_{-i} = \bar{\omega}_{-i}(\lambda_{-i}, w_i), \tag{31}$$

with $\overline{\omega}_{-i}(\lambda_{-i},w_i) = A_i(w_i)/\varepsilon\mathcal{S}(1-\gamma)\pi_{-i}$ where we set

$$A_i(w_i) = \varepsilon\frac{h_{-i}Q_{-i}}{2p_{-i}} + \varepsilon\frac{Q}{2} - \varepsilon\mathcal{S}w_i\frac{1-\gamma}{2}\pi_i - \varepsilon\mathcal{S}\frac{\tilde{c}_0 - r_0}{2} + \varepsilon c_{-i}(1-\gamma)\pi_{-i}\frac{\mathcal{S}}{2} - \frac{\lambda_{-i}}{2}.$$

The complementary slackness condition $\widetilde{p}_0(w_1, w_2) = p_2$ then eventually reduces to $\lambda_{-i} = \Lambda_{-i}(w_i)$ where

$$\Lambda_{-i}(w_i) = \varepsilon\mathcal{S}(1-\gamma)\pi_i w_i + \varepsilon\frac{h_{-i}Q_{-i}}{p_{-i}} + 3\varepsilon Q + \varepsilon\mathcal{S}(\tilde{c}_0 - r_0) + \varepsilon\mathcal{S}c_{-i}(1-\gamma)\pi_{-i} - 4\varepsilon\mathcal{S}p_2.$$

By inserting this expression of $\lambda_{-i} = \Lambda_{-i}(w_i)$ into the right-hand side of (31), we finally get $w_{-i} = \overline{\Omega}_{-i}(w_i)$ where

$$\overline{\Omega}_{-i}(w_i) = \frac{-(1-\gamma)\pi_i\mathcal{S}w_i - Q - \mathcal{S}(\tilde{c}_0 - r_0) + 2\mathcal{S}p_2}{\mathcal{S}(1-\gamma)\pi_{-i}}.$$

Now, consider the profit of leader MNO i given by

$$\mathcal{R}_i^{**}(w_i) = \mathcal{R}_i^*(w_1, w_2)_{|w_{-i}=\overline{\Omega}_{-i}(w_i)}.$$

The first derivative of $\mathcal{R}_i^{**}(w_i)$ with respect to w_i equals $d\mathcal{R}_i^{**}(w_i)/dw_i = \pi_i(1-\gamma)Q_0$ and is strictly positive; but in the present case, $w_i \in [0, w_i^0]$ where w_i^0 is either the abcissa or the ordinate of the intersection point of line δ with the axis $w_{-i} = 0$. The corresponding optimal unit price w_{-i}^* for the follower MNO $-i$ is therefore 0; as having no economic relevance for this partner MNO, this case (II) is eventually excluded.

B. *Proof of Proposition 3.*

We successively consider the two cases (a) $(w_1, w_2) \notin \Delta$ and (b) $(w_1, w_2) \in \Delta$.

First consider case (a). In view of (17), this corresponds to pairs (w_1, w_2) such that $p_0^*(w_1, w_2) = p_2$ and by (20), we easily show that $\mathcal{R}_i^*(w_1, w_2) = \mathcal{R}_i(p_2, w_i)$ is a linear and increasing function of w_i, $i \in \{1, 2\}$. The optimal value of $\mathcal{R}_i^*$ is thus obtained at $w_i = +\infty$; as this unbounded price gives the value $-\infty$ for $\mathcal{R}_0$, case (a) can therefore be excluded.

Now consider case (b). In view of (17), this corresponds to values of (w_1, w_2) such that $p_0^*(w_1, w_2) = \widetilde{p}_0(w_1, w_2)$. From definition (20) and the expression (18) of $\widetilde{p}_0(w_1, w_2)$, each function $\mathcal{R}_1^*(w_1, w_2)$ and $\mathcal{R}_2^*(w_1, w_2)$ is again expressed as a quadratic function of variable w_1 and w_2, respectively. The KKT conditions associated with optimization problem (24) for MNO 1, and optimization problem (25) for MNO 2, read

$$\begin{cases} \dfrac{\partial\mathcal{R}_i^*}{\partial w_i}(w_1, w_2) = \lambda_i\dfrac{(1-\gamma)\pi_i}{2}, \\ \lambda_i \geqslant 0, \qquad \lambda_i(\widetilde{p}_0(w_1, w_2) - p_2) = 0, \\ \widetilde{p}_0(w_1, w_2) \leqslant p_2; \end{cases} \tag{32}$$

for $i = 1$ and $i = 2$, respectively. At this point, we need to distinguish three cases according to the values of the Lagrange multipliers (I) $\lambda_1 = 0$ and $\lambda_2 = 0$, (II) $\lambda_1 > 0$ and $\lambda_2 > 0$, (III) $\lambda_1 = 0$ and $\lambda_2 > 0$ (or reversely $\lambda_1 > 0$ and $\lambda_2 = 0$).

(I) First assume $\lambda_1 = 0$ and $\lambda_2 = 0$. The simultaneous conditions

$$\frac{\partial\mathcal{R}_1^*}{\partial w_1}(w_1, w_2) = 0, \quad \frac{\partial\mathcal{R}_2^*}{\partial w_2}(w_1, w_2) = 0$$

yield the critical pair $\mathbf{w}^C \in D^A \cap D^B$, intersection point of lines D^A and D^B defined in (29). For each $i \in \{1, 2\}$, the second derivative $\partial^2\mathcal{R}_i^*(w_1, w_2)/\partial w_i^2$ equals the negative constant $-\varepsilon(1-\gamma)^2\pi_i^2\mathcal{S}$; $\mathcal{R}_i^*$ is thus a strictly concave function of the variable w_i and the point $\mathbf{w}^C$ is therefore the unique joint maximum for $\mathcal{R}_1^*$ and $\mathcal{R}_2^*$. Besides, it is readily shown that $\widetilde{p}_0(w_1^C, w_2^C) < p_2$ if and only if $r_0 > r_0^\flat$; in such a case, we then have $\mathbf{w}^C \in \Delta \setminus \delta$ and this point $\mathbf{w}^C$ is the optimal pair of wholesale prices.

(II) Now assume $\lambda_1 > 0$ and $\lambda_2 > 0$. Solving each KKT system (32) for $i = 1$ and $i = 2$, we get

$$\begin{cases} w_1(\lambda_1) = \dfrac{\dfrac{h_1Q_1}{p_1} + c_1(1-\gamma)\pi_1\mathcal{S} + 2Q - 2\mathcal{S}p_2 - \dfrac{\lambda_1}{\varepsilon}}{(1-\gamma)\pi_1\mathcal{S}}, \\ w_2(\lambda_1) = \dfrac{\dfrac{h_2Q_2}{p_2} + c_2(1-\gamma)\pi_2\mathcal{S} + 2Q - 2\mathcal{S}p_2 - \dfrac{\lambda_2}{\varepsilon}}{(1-\gamma)\pi_2\mathcal{S}}, \\ \lambda_1 + \lambda_2 = \varepsilon\mathcal{S}(r_0^\flat - r_0). \end{cases}$$

We have $\lambda_1 + \lambda_2 > 0$ if and only if $r_0 < r_0^\flat$. Calculating the respective values of $\varphi_1(\lambda_1) = \mathcal{R}_1^*(w_1(\lambda_1), w_2(\lambda_1))$ and $\varphi_2(\lambda_1) = \mathcal{R}_2^*(w_1(\lambda_1), w_2(\lambda_1))$ easily shows that φ_1 and φ_2 are linear functions over interval $[0, \varepsilon\mathcal{S}(r_0^\flat - r_0)]$ with respective maximum at $\lambda_1 = 0$ and $\lambda_1 = \varepsilon\mathcal{S}(r_0^\flat - r_0)$ (see Fig. 3). Consequently, there cannot be a joint solution that maximizes both $\mathcal{R}_1^*(w_1, w_2)$ and $\mathcal{R}_2^*(w_1, w_2)$ unless $r_0 = r_0^\flat$, in which case the optimal point coincides with $\mathbf{w}^C$.

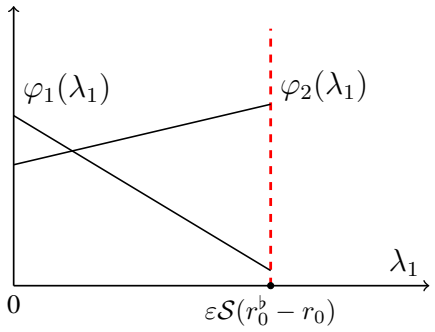


Fig. 3. Variations of linear functions φ_1 and φ_2 on interval $[0, \varepsilon\mathcal{S}(r_0^\flat - r_0)]$.

(III) Finally assume $\lambda_1 = 0$ and $\lambda_2 > 0$. Solving then each KKT system (32) for $i = 1$ and $i = 2$ yields the critical pair $\mathbf{w}^A$ and the value $\lambda_2 = \varepsilon\mathcal{S}(r_0^\flat - r_0)$; in particular, we have $\lambda_2 > 0$ if and only if $r_0 < r_0^\flat$. Therefore, $\mathbf{w}^A$ is the pair of optimal wholesale prices when $r_0 < r_0^\flat$. Symmetrically, the case $\lambda_1 > 0$ and $\lambda_2 = 0$ gives the optimal pair $\mathbf{w}^B$ if $r_0 < r_0^\flat$. In view of the above properties of functions φ_1 and φ_2, however, we easily show that neither $\mathbf{w}^A$ nor $\mathbf{w}^B$ can be a joint solution that simultaneously maximizes $\mathcal{R}_1^*(w_1, w_2)$ and $\mathcal{R}_2^*(w_1, w_2)$, as required in (24) and (25). This joint problem has consequently no solution when $r_0 < r_0^\flat$.

HyLine: a Simple and Practical Flow Scheduling for Commodity Datacenters

Soheil Abbasloo, Yang Xu, H. Jonathan Chao ({ab.soheil, yang, chao}@nyu.edu)
New York University

***Abstract*— Today's datacenter networks (DCNs) have been built upon multipath topologies where each path contains multiple links. However, flow scheduling schemes proposed to minimize flow completion times (FCT) in DCNs are based on algorithms which are optimum or close-to-optimum only over single link. Moreover, most of these scheduling schemes seek either fully centralized approaches having overhead of communicating to a central entity or fully distributed approaches requiring changes in the fabric.**

Motivated by these shortcomings, we present HyLine a simple scheduling design for commodity DCNs which is equipped with a joint load-balancing and flow scheduling (path-aware) design exploiting the multipath nature of DCNs. HyLine takes a hybrid approach and uses the global-awareness of centralized and agility of distributed techniques without requiring any changes in the fabric. To that end, it determines a threshold margin identifying flows for which using centralized approach is beneficial.

We have shown through extensive ns2 simulations that despite HyLine's simplicity, it significantly outperforms existing schemes and achieves lower average and 99th percentile FCTs. For instance, compared to Qjump–state-of-the-art practical scheme– and pFabric–one of the best performing flow scheduling schemes– HyLine reduces average FCT up to 68% and 31%, respectively, under a production datacenter workload.

I. INTRODUCTION

User satisfaction (and total revenue) of today's popular datacenter applications such as search, social networks, and recommendation systems is closely related to the response times of these interactive applications. This motivates recent research to propose new datacenter (DC) transport designs for minimizing average flow completion times (AFCT) as the primary objective that is mainly determined by the end-to-end latency of datacenter networks (DCNs).

Prioritization is one of the main techniques used by different approaches to achieve lower AFCTs [1-5]. Wide range of these proposals use shortest remaining processing time (SRPT) (or its simplified versions), the optimum scheduling algorithm when used over a single link [1], to minimize AFCT in DCNs. However, as we show in section III, these algorithms are suboptimal for minimizing AFCT when each path in the network has multiple links. This issue will be escalated when multipath nature of today's DCNs is considered.

Agility of fully in-network schemes motivates some proposals to keep all changes in the network to achieve lower response times [1, 6, 2]. However, this usually requires changes in the fabric which brings extra costs for the datacenter owners [18, 7]. On the other hand, using centralized schemes such as [8], in which fabric will not be modified, comes at cost of performance degradation due to the delay introduced by the controller. This will be escalated when it is considered that most of the DC flows are very small and can be finished in just a few round trip times (RTTs) [9, 7]. Moreover, using explicit rate control mechanisms to precisely adjust flows' rates in the network leads to high complexity in the centralized approach (e.g., [3]) or the need to modify switches to coordinate with each other for finding and maintaining the best rates in the distributed approach (e.g., [6, 2]).

To overcome these shortcomings, in this paper, we present HyLine, a simple and practical flow scheduling design which:

1. Takes a hybrid approach requiring no changes in the fabric, and uses both global-awareness of centralized and agility of distributed techniques such as priority flow control (PFC) in layer 2,

2. Uses a joint load-balancing and flow scheduling (path-aware scheduling) policy to exploit the multipath nature of DCNs, and

3. Does not use any complicated per flow rate adjustment mechanism.

To that end, HyLine determines a threshold identifying 2 categories of flows: flows that should be scheduled in a centralized manner (2nd class flows having sizes larger than the threshold) and flows that should not be (1st class flows having size smaller than the threshold). Having that threshold, end-hosts simply assign 1st class flows to the higher priority queue in commodity switches and send them to the network at line rate (TCP handles any further required rate adjustment). 2nd class flows that are assigned to the lower priority queue will be scheduled before coming to the network. Each of the 2nd class flows should first send a request including flows' information to HyLine's central MANager (MAN) seeking its permission. MAN is responsible to control 2nd class flows in a very simple stop-and-go fashion. To do that, it uses simple path-aware scheduling policy to find the best path for the requested flow based on flow's information (priority). If a path is found for the new flow, MAN sends back a Go signal carrying the path that should be used by the corresponding flow. All permitted 2nd class flows enter the network at their end-host's line rate using the assigned paths (each edge-link will be used by at most one 2nd class flow at a time). MAN also sends a Stop signal to the preempted flows or the ones that cannot be served yet.

ISBN 978-3-903176-08-9

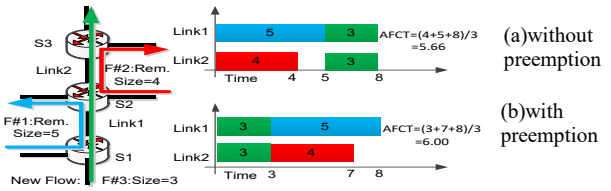


Fig. 1. AFCT with and without preemption.

We evaluate HyLine's performance through extensive packet-level simulations in ns2 [10]. The results show that despite simple nature of HyLine's design, it significantly outperforms recent schemes including pFabric [1], one of the best performing flow scheduling schemes, Qjump [5], the state-of-the-art practical scheme, and DCTCP [9]. In particular, compared to pFabric, Qjump, and DCTCP, HyLine reduces AFCT up to 31%, 68%, and 88% respectively, under a realistic DCN workload [9].

II. RELATED WORKS

Transport Designs: There are vast number of TCP designs targeting a specific environment (e.g., [28] in cellular context and [3] in DCN context). Most of recent TCP proposals in DCN context use various prioritization mechanisms to minimize FCT [1, 4, 2, 3]. For instance, they assign different rates to flows based on their criticality [2], tag each packet with its corresponding priority and serve it regarding that priority in the network [1], use strict priority scheduling among queues in switches and assign flows dynamically to different levels of priority [4], or use a combination of these strategies [3]. Although designs that use the prioritization idea achieve good performance, they all are based on single-path scheduling algorithms such as SRPT. Therefore, some of these schemes (e.g., [3, 24]) only test their designs in single-path scenarios. Most of the other ones including [1, 4, 11, 5] use packet spraying [12] as load balancing mechanism to run their schemes on a multipath DCN. However, packet spraying is not an available feature in most of the commodity switches and is not used in commodity DCNs [18, 7, 13]. Therefore, we avoid using such load balancing mechanisms in this paper, though they might lead to good performance.

Joint Transport-Load Balancing Designs: Almost none of the load-balancing designs in the network layer are priority-aware. To the best of our knowledge, there is only one scheme called DeTail [14] in which a cross-layer approach is used to reduce the long tail of FCTs in DCNs. Although DeTail achieves good performance, a lack of backward compatibility and the need for changing both switches and end-hosts make it very hard if not impractical for commodity DCNs. Fastpass [8] uses a centralized entity to handle not only scheduling block but also load balancing block. However, it also follows the traditional approach of designing scheduling block (timeslot allocation block in [8]) and load balancing block (path selection block in [8]) separately. Moreover, Fastpass could not minimize FCTs, because at least for the very small flows that could be finished in a few RTTs, it adds (at least) one RTT delay caused by communication with Fastpass's central controller.

Load Balancing Designs: Nearly all load balancing schemes in DCNs are designed based on the fairness nature of the network among all flows [15, 16, 17]. For instance, Hedera [15] detects flows with sizes more than 100MB (10% of the link's capacity) and estimates their demands based on max-min fairness criterion to reroute them. However, as recent transport designs show, minimizing FCT in DCNs should be done through considering the prioritization in the network. Therefore, following the fairness criterion for designing the load balancing block will cause suboptimal FCT, though a better load balancing design, such as [16, 17], could reduce the overall FCT.

III. MOTIVATIONS & DESIGN DECISIONS

Scheduling Over Single-Link vs. Multiple-Link Paths: It is usually mentioned in the literature that preempting lower priority flows to serve higher priority ones minimizes the AFCT. This statement is a direct result of considering SRPT–the optimum solution when scheduling over a single link–as main algorithm to schedule flows (e.g., [1, 2, 3]). However, we show that this statement is wrong in a network where paths contain multiple links. For that purpose, we use a simple example shown in Fig. 1 where flows #1 and #2 have 5 and 4 remaining units respectively. Now a new flow (Flow #3) with 3 units comes to the network (consider remaining size of each flow as its priority i.e., smaller size has higher priority). So clearly, in contrast with SRPT, using no preemption (Fig. 1.a) leads to smaller AFCT. This is important to mention that using either local-aware SRPT (in S1 and S2 switches) (as in [1]) or global-aware SRPT (as in [3]) will lead to the suboptimal result (Fig. 1.b). Therefore, the incorrectness of the mentioned statement illustrates the need for designing better scheduling algorithms by considering the multiple-link nature of paths in DCNs.

Simple, deployable, and end-to-end: Datacenter owners usually prefer using scale out (using commodity switches) to scale up (using high-end switches with high-end new features) to build their networks [18, 7]. This motivates us to not modify any switches in the network, though modifying switches might give good performance [1, 16, 6, 24] and look for a simple end-to-end solution which is deployment friendly.

Why Hybrid? Centralized approaches are attractive because they could use global knowledge of the network to make better decisions [15, 8]. However, they suffer from some issues. Due to the communication delay with the controller, scheduling small flows (most flows in DCNs [9]) through centralized approaches is not desired. Another issue is their response times. For instance, the scheduler in [15] runs every 5 seconds, which leads to its bad performance compared to distributed solutions such as [16]. For centralized schemes such as [8] that require highly synchronized nodes, synchronization is another issue. Keeping nodes synchronized at the order of one microsecond as [8] requires, is challenging in a real DC environment [5]. On the other hand, although responsiveness of distributed approaches [16, 6, 2] is good, they require adding new functionalities to the switches. Therefore, instead of using a fully distributed or a fully centralized technique, it is beneficial to come up with a hybrid approach combining the global awareness of centralized techniques and the agility of distributed ones.

Why Path-Aware Scheduling? One of our main ideas is that load balancing and flow scheduling are dependent design

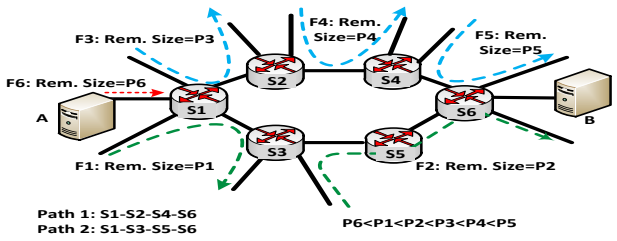


Fig. 2. A simple multipath network.

blocks in DCNs and should be designed together to achieve a global objective such as minimizing AFCT in a multipath DCN. So, instead of using single-path scheduling policies (e.g., [1, 4, 6, 5, 2, 3, 24]), we consider a path-aware scheduling logic.

IV. DESIGN

Scheduling flows to minimize AFCT in single path scenario is an NP-hard problem [1]. This problem in multipath scenarios will remain NP-hard. In this section, we introduce the key design principles of HyLine, which uses heuristic approach to minimize AFCT using path-aware scheduling in multipath commodity DCNs.

A. HyLine's Big Picture

End-Hosts: In HyLine, end-hosts are responsible for classifying all flows into two classes: 1) Latency-sensitive flows, i.e., the small flows, that require less queuing and transmission delays. 2) Bandwidth hungry flows that could tolerate some delays during their transmission. This classification will be done using a threshold provided by MAN, a logically centralized network manager. All of the flows in the latency-sensitive class (1st class i.e., flows having sizes smaller than threshold) are assigned to the higher priority queue in switches (Q1) and all of the bandwidth hungry flows (2nd class i.e., flows having sizes bigger than threshold) are assigned to the lower priority queue in switches (Q2). Next, all 1st class flows are sent to the network at line rate, and flow-based ECMP is used for balancing their loads among available paths. However, end-hosts should first send a Request to Send (RTS) message to MAN asking permission before sending any of their 2nd class flows to the network. This RTS carries source, destination, and size of the flow.

MAN: MAN is the logically centralized entity in HyLine that is responsible for scheduling 2nd class flows. To this end, it guides transmission of all of the 2nd class flows in a very simple Stop-and-Go fashion. If MAN decides that a flow could come to the network, it sends back a Clear to Send (CTS) message (i.e., Go) carrying the path that should be used for transmission of this flow. If not, it sends back a Stop to Send (STS) message forcing the flow to be kept at the edge of network. Flows that get CTS messages are sent to the network at line rate. These permitted flows only would be stopped momentarily in two conditions by two different mechanisms:

First: When there is no more bandwidth available to serve a new incoming 2nd class flow with higher priority than a few of the permitted ones. In this case, MAN uses a path-aware preemption mechanism (§4.3) to select the best set of flows to preempt and sends the STS messages to the preempted ones and stops them.

Second: When permitted 2nd class flows are going to be dropped at switches due to a high load in higher priority queue caused by 1st class flows. In this case, to keep the design simple and practical, instead of using fine-grained monitoring of the queue occupancies for each switch, PFC– defined as part of IEEE 802.1Qbb standard [19] and an available feature in today's commodity switches [9, 20]–is used to pause permitted 2nd class flows without any need for coordination with MAN.

When a 2nd class flow is finished (or close to being finished), its corresponding end-host sends a FIN message to MAN indicating that the path (and bandwidth) allocated for this flow is now free. Then, MAN assigns the available resources to other flows which are stopped (by MAN).

B. Why it works?

There are three main reasons why HyLine boosts performance of latency sensitive flows in DCNs:

1) Queue length builds up in a DCN mainly as a result of having bandwidth hungry flows. This class of flows occupies queues and causes dramatic increase in completion times of small flows due to increasing buffer delay and increasing drop rate of small flows' packets and the consequent retransmission of them. Therefore, giving credit to small flows and allowing them to be served first in the switches significantly reduces their completion times.

2) Due to the hash-based nature of flow-based ECMP, this load balancing scheme performs very well when it is used for a network that consists of only small flows [16].

3) Making the bandwidth hungry flows (large portion of all bytes transferred in DCN [9, 7]) to be served after serving the 1st class flows opens room for the 1st class flows to bypass the slow start phase of TCP and finish as soon as possible.

In addition, HyLine boosts performance of bandwidth hungry flows, i.e., the 2nd class compared to single-path based flow scheduler proposals [9, 1, 4, 5], because:

1) Using the MAN, a logically centralized network manager, enables HyLine to have global knowledge of the network for scheduling the 2nd class flows.

2) HyLine benefits from the pre-planned nature of DCN topologies and uses a preemption policy that not only considers flows' information but also network's topology information at the time of scheduling.

3) Since HyLine pushes back and stops the 2nd class flows at the edge of the network when network could not serve them at the current time, packet drops, retransmissions, queue occupancy, and congestion for the 2nd class flows are reduced dramatically.

C. Path-Aware Flow Scheduling Heuristic

In this section, we introduce a new path-aware scheduling policy used in the core of HyLine by considering multiple-path DCNs where each path has multiple links.

To Preempt or Not to Preempt: To explain the HyLine's path-aware scheduling policy, we use the example shown in Fig. 2. Flow #6 (F6) with size p_6 is generated at A and destined to B, while there is no enough bandwidth to serve this flow without preempting others (different links might contain different flows, but Fig. 2 only shows the ones that have lower priorities (higher remaining sizes) than F6). Similar to the example in Fig. 1, total flow completion time (TF) when using each path can be calculated as follow:

Without Preemption:

$$\begin{cases} TF_{Path1} = [p_1 + p_2] + [p_3 + p_4 + p_5 + (p_6 + p_5)] = \sum_1^6 p_i + p_5 \\ TF_{Path2} = [p_1 + p_2 + (p_6 + p_2)] + [p_3 + p_4 + p_5] = \sum_1^6 p_i + p_2 \end{cases} \quad (1)$$

With Preemption:

$$\begin{cases} TF_{path1} = [p_1 + p_2] + [(p_6 + p_3) + (p_6 + p_4) + (p_6 + p_5) + p_6] \\ \qquad = \sum_1^6 p_i + 3p_6 \\ TF_{Path2} = [(p_1 + p_6) + (p_2 + p_6) + p_6] + [p_3 + p_4 + p_5] \\ \qquad = \sum_1^6 p_i + 2p_6 \end{cases} \quad (2)$$

As these equations illustrate, path 2 is the best choice, and if $2p_6 < p_2$, preemption should be used.

In general, when N, P_{max} , and P_{new} represent number of required preemption on a path, maximum priority on a path, and priority of the new flow, if $N \times P_{new} < P_{max}$, preemption is preferred, while in other cases, using no-preemption leads to smaller AFCT. Therefore, totally, the path that has the Minimum of either $N \times P_{new}$ (in short, MNP) or P_{max} is the best path.

D. Scheduling Logic's Details

HyLine's main path-aware scheduling policy is based on the fact that permitted flows are sent at edge link's line rate. This makes the overall design very simple and omits the need for any precise rate calculation and sophisticated scheduling policies. Another key rule to simplify the logic and reduce the time complexity is out-of-order delivery avoidance. To avoid out-of-order delivery, paths allocated for permitted flows could not be changed. In other words, only new flows and already stopped ones (by MAN) could be assigned to other paths.

Algorithm 1 shows MAN's main logic. With new incoming (RTS) request for a flow, MAN looks for the best path for the new flow. For this purpose, MAN finds the number of required preemptions and lowest priority on each path.

Balanced Load: When new flow is permitted to come to the network, and there are multiple choices for the final path, the remaining BW of these paths is considered and the path with the maximum remaining BW is selected for the new incoming flow (Remaining BW of a path is defined as the minimum remaining BW of the links in that path). If the remaining BW is also equal for those paths, random selection will be used to break the tie. MAN will only consider 2nd class flows to calculate remaining BW, because it does not have any information about 1st class flows. HyLine manages the impact of 1st class flows by using PFC in the network.

Algorithm 1: HyLine's Main Algorithm

```
1  Function NewRequest(f)
2      FlowList ⟵ f /* insert to the sorted list */
3      Schedule(f);
4  Function RemoveRequest(f)
5      RemoveFlowfromPath(f.path,f)
6      FlowList ⟶ f /* remove f from list */
7      ReSchedule();
8  Function Schedule(f)
9      [found, premptList, path] = FindPath(f)
10     if found then
11         for Flow ∈ premptList do
12             RemoveFlowfromPath(path, Flow)
13             SendSTS(Flow.src)
14         AddFlowtoPath(path,f)
15         SendCTS(f.src, path)
16         f.IsStopped = false; ReSchedule()
17         for Flow ∈ premptList do
18             Flow.IsStopped = true
19     else
20         SendSTS(f.src)
21 Function ReSchedule
22     for f ∈ Flowlist && if(f.IsStopped) do
23         (found, premptList, path) = FindPath(f)
24         if found then
25             for Flow ∈ preemptList do
26                 RemoveFlowfromPath(path, Flow)
27                 SendSTS(Flow.src)
28                 Flow.IsStopped = true
29             AddFlowtoPath(path, f)
30             SendCTS(f.src, path)
31             Flow.IsStopped = false
32 Function FindPath(f)
33     mnp = ∞; bw = 0; pathList = void;
34     MinMaxPrio = ∞; preemptionList = void
35     for p ∈ availablepaths(f.src,f.dst) do
36         MinMaxPrio =
             UpdateMinMaxPrio(p,f, MinMaxPrio)
37         found = findMNP(p,f)
38         if found then
39             if mnp > p.mnp & p.remBw > bw then
40                 pathList.clear(); pathList.insert(p)
41                 mnp = p.mnp; bw = p.bw
42             else if mnp == p.mnp & p.remBw == bw then
43                 pathList.insert(p)
44     if pathList.size ≥ 1 then
45         found = true; path = pathList.Random()
46         preemptList = p.preemptList
47     if found & mnp × f.priority ≥ MinMaxPrio then
48         found = false
49     return found, premptList, path
```

Reschedule: After selecting a path for a new incoming flow and likely stopping/preempting some other flows on this path, there might be available room for flows that have been stopped before. Therefore, in case of preemption, MAN checks the possibility of admitting more flows into the network (considering out-of-order delivery avoidance rule). Clearly, there is a trade-off between adding more rounds of rescheduling to admit more probable flows and the overall time complexity of the algorithm. To reduce the time complexity of the main logic, we decided to do only one round of rescheduling. The results in §5 show that this decision still leads to very good overall performance.

HyLine's Time Complexity: Here, we show that the time complexity of HyLine is O(|F|) where |F| is the total number of active flows in the 2nd class. To show this result, we first should notice that the maximum number of permitted flows on a link has an upper bound that is independent of the number of flows

considering the assumption that all flows are sent at line rate. Assuming that the lowest and highest link rates on a path are S bps and M bps, respectively, the maximum number of 2nd class flows in a link of that path is M/S. When findMNP procedure (line 37 in Algorithm 1) is implemented simply by exploring the entire valid preemption list of flows in a path, for each path, at most, it looks at $(M/S)^l$ combinations in which, l is number of links in a path. For instance, in a 3-tier datacenter, l is equal to 6. Therefore, FindPath takes constant time. In addition, the out-of-order delivery avoidance rule causes a flow to be considered during the ReSchedule procedure at most once. This illustrates that the Complexity of Schedule function, which is equal to the total complexity of the algorithm, is O(|F|).

PFC and Head-of-Line Blocking Issue: PFC if used in a normal network will cause head-of-line blocking issue for flows using the same priority queue. However, in HyLine, MAN pushes most of the 2nd class flows back and stops them from coming into the network. This strategy significantly reduces the head-of-line blocking issue and as the results in §5.5 show, using PFC boosts the overall performance.

Rate Control: HyLine has no complicated rate control mechanism. It uses TCP, and to send flows at line rate, changes initial congestion window size. However, 2nd class flows being stopped by MAN should not cause TCP time-outs. So we modify TCP to avoid such time-outs for the 2nd class flows (when they receive STS signal from MAN) without affecting TCP time-out mechanism for 1st class flows. This modification only requires adding a few lines of code to the original TCP implementation.

E. The Threshold to Distinguish Classes of Traffic

Some recent studies [1, 4] tried to formalize the problem of finding optimum thresholds to distinguish different flows based on their sizes and use available priority queues in today's commodity switches to separate packets of different types of flows. PIAS and pFabric use simple M/M/1 and M/G/1 queue models, respectively, to find the best threshold values. Even with these simplifications, the problem of finding optimum thresholds is complicated [1] and NP-hard [4]. Moreover, these simplifications do not work in our case. In fact, none of the M/M/1 or M/G/1 FIFO queue models are valid approximations for our 2nd queue. Even in simple single queue scenario where our 2nd queue scheduling mechanism is equal to SRPT, FIFO queue models should be replaced by complex SRPT queue models [21]. From this point of view, the problem of finding the best thresholds becomes even more complicated than before. Therefore, in this paper, we choose another direction and instead of finding the optimum threshold, we determine a band for practical threshold values.

Lower Bound: In theory, forcing more flows to be controlled by MAN (i.e., decreasing the threshold) increases the performance because of having a global view of the network during the scheduling; however, in practice, reducing threshold (H) causes additional delays for the small flows due to the controller's delay (both network delay for reaching the MAN and computation delay of MAN). To simplify the analysis and find a lower bound for H, we consider single queue model and use mean queue analysis. We define a delay cost, T_{cost}, for any flow which is controlled by MAN, f_s as the smallest flow in the 2nd queue (f_s's size = H), and W_{f_s} as expected waiting time of f_s (time from when it first arrives to when it receives service for the first time). We argue that W_{f_s} should not be smaller than T_{cost} (if $T_{cost} > W_{f_s}$, putting f_s in the 1st queue (increasing the threshold to $H + \varepsilon$) will cause lower FCT for f_s).

The f_s will be served only after serving all flows in the 1st queue and after serving all flows having smaller remaining sizes (but originally bigger) than it in the 2nd queue. In other words, any flow with smaller size than f_s in 1st queue or any flow with smaller remaining size than f_s in 2nd queue will preempt f_s before it receives service for the first time. So, from f_s's point of view, W_{f_s} in this network is equal to W_{f_s} in a network where there is only a single SRPT queue. Therefore, we can use [21]'s analysis for an M/G/1/SRPT queue to find average value of W_{f_s}:

$$E[W(x)] = \frac{\lambda\left(m_2(x)+x^2\left(1-F(x)\right)\right)}{2\left(1-\rho(x)\right)^2} \quad (3)$$

Here, we denote average arrival rate by λ, service time (service time=size/service rate, service rate=link speed) of a flow by X, CDF of service time distribution by $F(x)$, $m_2(x) = \int_0^x t^2 f(t)dt$, and the load made up by the flows of service time less than or equal x by $\rho(x) = \lambda\overline{X_x}$ in which $\overline{X_x} = \int_0^x tf(t)dt$. Substituting x in this formula with $h = H/C$ in which C represents bandwidth of the link, will lead to calculation of $E[W(h)] = E\left[W_{f_s}\right]$. So, the following inequality represents the lower bound:

$$T_{cost} \leq \frac{\lambda\left(m_2(h)+h^2\left(1-F(h)\right)\right)}{2\left(1-\rho(h)\right)^2} \quad (4)$$

Upper Bound: Increasing H puts more flows into the 1st class, causes congestion in the 1st queue and consequently decreases the performance. Therefore, to address this issue, we require an upper bound on H. Here, the important observation is that almost all of the schemes including normal TCP perform very well when load is very low (less than 10%) [1, 4, 3, 5]. The reason is that at low load, inter-arrival of the flows is large enough to serve flows without having congestion issue. Based on this important observation, we cap the overall load of the 1st queue. In more detail, we choose $\rho_1 = \rho(h) \leq \frac{\rho_{total}}{10}$. Since $\rho_{total} < 1$, this choice guarantees that the total load in the 1st queue (ρ_1) is always smaller than 10%; therefore, congestion in the 1st queue will not be an issue. So, following equation represents the upper bound:

$$\frac{\rho_1}{\rho_{total}} = \frac{\overline{X_h}}{\overline{X_{total}}} \leq 0.1 \quad (5)$$

$E[W(H)]$ and $\frac{\rho_1}{\rho_{total}}$ for web search workload and different loads (up to 90%) are shown in Fig. 3 (C=1Gbps). The band for choosing H in a moderate load of 60% is depicted in this figure too (through this paper we assume $T_{cost} = 100\mu s$).

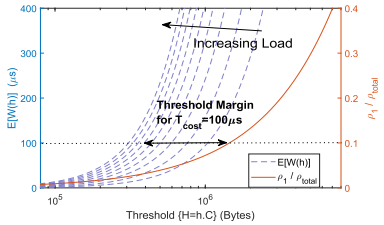


Fig. 3. $\boldsymbol{E[W(H)]}$ and $\boldsymbol{\rho_1/\rho_{total}}$

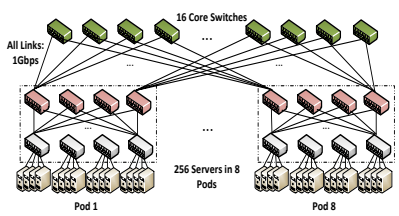
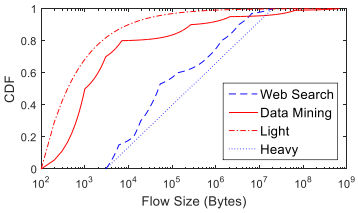


Fig. 4. Simulation setup (a) The 3-tier topology



(b) Flow size distributions of workloads used

Workload	% of Flows smaller than 100KB
Heavy	40%
Web Search	58%
Data Mining	83%
Light	97%

Static vs. Dynamic Threshold Assignment: Clearly, assigning thresholds dynamically based on the load of the flows (as lower bound criterion suggests) is beneficial. For that purpose, different agents at end-hosts could periodically report summary of all their flows to the MAN. Later, MAN will use these reports to choose the threshold and report it back to the end-hosts. Although HyLine's structure enables us to use this approach, to keep the design simple and practical we use a static threshold assignment, and in §5, we will show that this approach works very well for different loads and even for different types of workloads. So, through the rest of this paper, based on Fig. 3, we choose H=1MB.

V. EVALUATION

In this section, we evaluate the performance of HyLine using extensive packet-level simulations in ns2 (available at: https://github.com/soheil-ab/hyline). First, we compare the performance of HyLine with existing proposals including Qjump [5], pFabric [1], DCTCP [9], and TCP-New Reno. Then, through micro-benchmarks, we evaluate HyLine's performance such as its sensitivity to the threshold value, improvements caused by PFC.

A. Simulation Settings

Datacenter Topology: We use a 3-tier fat-tree topology [18] which is the base topology for today's DCNs [22,13] for our evaluation (Fig. 4.a). The topology includes 8 pods interconnecting 256 end-hosts using 80 8-port switches with a 300µs overall end-to-end RTT delay between end-hosts located in different pods.

Load-balancing Mechanism: To have a fair comparison of HyLine's performance with other single-path based flow-scheduling schemes, we use flow-based ECMP used in commodity DCNs [18, 7] as the load balancing scheme.

Traffic Workloads: We use two realistic workloads from production datacenters: web search workload [9] and data mining workload [7]. In addition, we use 2 other synthetic workloads named Heavy and Light to change the heavy-tailedness of the traffic and do stress tests. The flow size distributions of all workloads are shown in Fig. 4.b.

Performance Metrics: We consider AFCT and 99th percentile FCT of flows as the performance objectives like prior work [1, 4, 2, 3]. We normalize all FCTs to the flows' ideal values achieved if each flow is transmitted over the fabric without any interference from competing traffic. In addition, since most of the datacenter applications (from search and social networks to MapReduce) use partition-aggregate structure equipped with different deadlines for flows in different layers of its hierarchy [23], similar to prior work [1, 3, 23], we use the application throughput defined as the fraction of flows that meet their deadline as another performance metric to investigate the impact of HyLine on real applications.

Schemes Compared: We compare HyLine with Qjump [5], pFabric [1], DCTCP [9], and TCP-New Reno with Sack. The parameters used for the evaluation of these schemes are selected based on their authors' recommendations or reflect the best settings that we have experimentally determined (Table 1). We use these parameters for evaluations in this section unless otherwise specified.

PFC Implementation in ns2: We use a simplified version of PFC (on/off style) that we have added to ns2 simulator. For that purpose, when the queue size hits a threshold (pause threshold), the switch sends pause signal to upstream switch. When the queue size becomes less than another threshold (resume threshold) the switch sends resume message.

B. Overall Performance

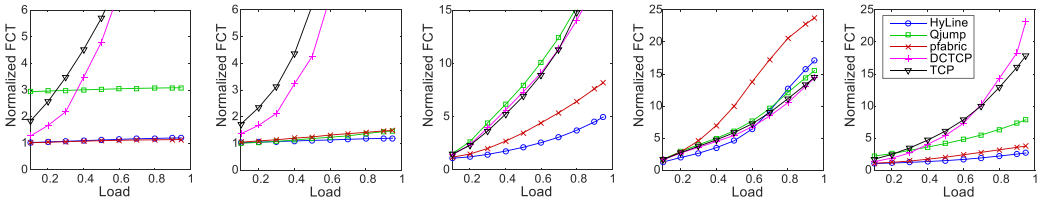
In this section, we present the overall performance of HyLine under the aforementioned workload and DCN topology. [9, 7]. We show that despite HyLine's simplicity, it outperforms all compared schemes.

Overall AFCT: The overall normalized FCT of flows with different schemes for search and data mining workloads are shown in Fig. 5.e and Fig. 6.e, respectively. As these results illustrate, HyLine achieves the best performance among all compared schemes. For instance, AFCT using HyLine is ~3-31% and ~52-66% lower than pFabric and Qjump respectively. All schemes generally perform better in data mining workload. The reason is that in this workload probability of having two large flows competing for the same link is less than search workload (Fig. 4.b). For this workload, HyLine achieves ~18-30% lower AFCT than Qjump and compared to pFabric performs roughly the same.

AFCT in More Detail: As expected, pFabric performs well for the very small flows in (0, 100kB] range (Fig. 5.a and Fig.

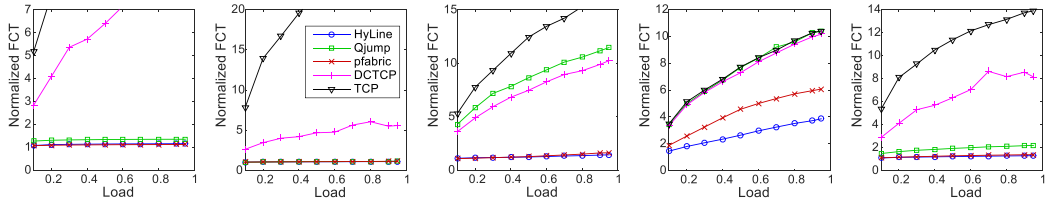
TABLE I. DEFAULT SIMULATION SETTINGS

Scheme	*Parameters*
pFabric	qsize = 50pkts (=2×BDP), initCwnd = 25pkts (=*BDP*), minRTO =1ms (≈3× *RTT*)
Qjump	qsize = 225pkts, initCwnd = 25pkts, minRTO = 4ms
dctcp & tcp	qsize = 225pkts, initCwnd = 10pkts, minRTO = 4ms
HyLine	qsize = 225pkts, initCwnd = 25pkts, H=1MB, Tcost=100us, minRTO = 4ms, initCwnd (2nd class) = 25pkts, minRTO (2nd class) = 1s, pause threshold=215pkts, resume threshold=205pkts



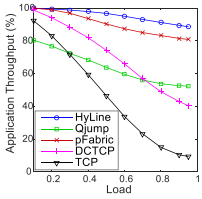
(a) (0,100kB] (b) (100kB, 1MB] (c) (1MB, 10MB] (d) (10MB, ∞) (e) Overall

Fig. 5. Normalized FCT statistics across different flow sizes for web search workload.

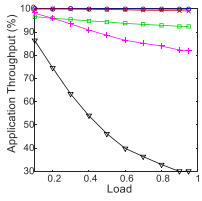


(a) (0,100kB] (b) (100kB, 1MB] (c) (1MB, 10MB] (d) (10MB, ∞) (e) Overall

Fig. 6. Normalized FCT statistics across different flow sizes for data mining workload.



(a) Web Search



(b) Data Mining

Fig. 7. Application Throughput across different loads

6.a). However, it comes at the expense of performance reduction for other ranges of flows, due to its local strategy of dropping packets at earliest stages of the network and reacting to this sooner by using small priority queues in switches and small timeouts at end-hosts. In contrast, HyLine allows the other 1st class packets (i.e. flows in (100kB, 1MB] range) to be queued in switches too. Considering multipath nature of network and the fact that all of these 1st class flows will not compete for the same output links in the next stages of the network, this increases the chance of serving flows in (100kB, 1MB] range later in the network (Fig. 5.b and Fig. 6.b). Moreover, Qjump cannot achieve very good performance for the small flows (specially for the search workload (Fig. 5.a)) because it reduces the throughput of these flows to give more bandwidth to the bigger ones.

For the 2nd class flows ((1MB, 10MB] and (10MB, ∞) ranges), HyLine benefits from having a global view and path-aware nature in its scheduler compared to other schemes . So, as Fig. 5.c-d and Fig. 6.c-d illustrate, it performs better than all other schemes for almost all loads and workloads except very high loads in search workload for flows in (10MB, ∞) range. For high loads in this range (Fig. 5.d), since total number of flows including big flows increases, the total number of preemptions for this range of flows increases too. Therefore, largest flows in the network face more preemption delay. In contrast, TCP achieves best performance at high loads (Fig. 5.d), because it loses less bandwidth due to the fairness nature of its design.

C. *Varying Performance Metrics*

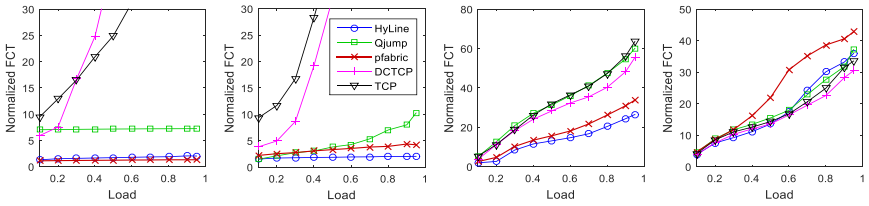
Application Throughput: Most of the today's datacenter applications use partition-aggregate structure in which flows in each level of the hierarchy have deadlines [23]. For instance, in a search application, if responses (flows) from workers miss their deadlines, they are not included in the total response, typically hurt the response quality, and waste network bandwidth. Therefore, to investigate impact of HyLine for such applications, we assign different deadlines to different flows and similar to prior work [1, 2, 23] consider application throughput as the performance metric. Here, deadline of each flow is considered 4x of its ideal completion time achieved when there are no other competing flows in the network. We used tighter and looser deadlines for flows too, but since the overall results are similar to the presented results, for brevity, we only report the results for the mentioned deadline. Fig. 7 depicts the overall results for two realistic workloads across different loads. HyLine outperforms other schemes for both workloads.

Since in both workloads, most traffic are small flows, finishing these small flows faster increases the probability of meeting their deadlines. Therefore, schemes which achieve better results for small flows potentially perform better for deadline-aware traffic too. That's why HyLine and pFabric perform very well compared to other schemes. It is important to notice that HyLine achieves this performance without any changes in the network, while pFabric requires changes in switches.

99th Percentile: In addition to previous metrics, we also consider the 99th percentile FCT as a performance metric to have a better comparison of HyLine with other schemes. Fig. 8 and Fig. 9 show the results of 99th percentile FCT for data mining and search workloads respectively for different flows' size ranges. 99th percentile result's pattern is similar to the AFCT result's pattern discussed earlier.

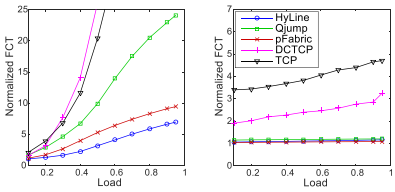
D. *Impact of Workload*

So far we evaluated HyLine under realistic DCN workloads. However, there might be still two concerns about the HyLine's performance: 1-What if traffic consists of more 1st class flows? 2-What if a workload consists of more 2nd class flows? To evaluate the performance under these two corner cases, we used Bounded-Pareto distribution to generate 2 synthetic workloads named Light and Heavy (Fig. 4.b). In Light workload, 97% of the flows are smaller than 100KB, while this number is only 40% for Heavy workload. This will provide us with workloads to check the two mentioned concerns. Fig. 10 and Fig. 11 show the AFCT, and application throughput results using Light and Heavy workloads. Under Light workload, all schemes generally



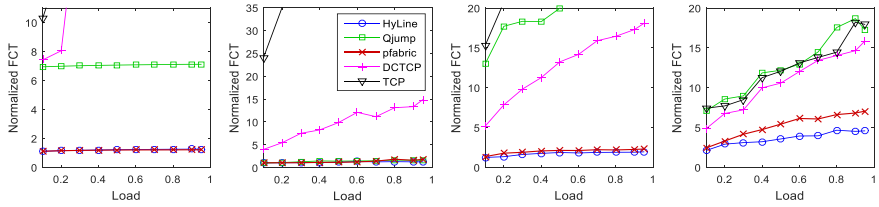
(a) (0,100kB] (b) (100kB, 1MB] (c) (1MB, 10MB] (d) (10MB, ∞)

Fig. 9. Normalized 99th percentile FCT statistics for web search workload across different flow sizes.



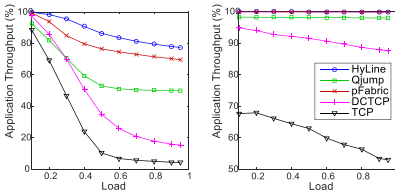
(a) Heavy workloads (b) Light workloads

Fig. 10. Normalized FCT statistics



(a) (0,100kB] (b) (100kB, 1MB] (c) (1MB, 10MB] (d) (10MB, ∞)

Fig. 8. Normalized 99th percentile FCT statistics for data mining workload across different flow sizes.



(a) Heavy workload (b) Light Workload

Fig. 11. Application Throughput

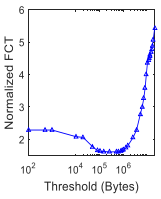
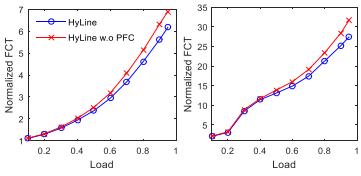


Fig. 12. AFCT across different thresholds.



(a) AFCT (b) 99th Percentile FCT

Fig. 13. Performance with & without PFC

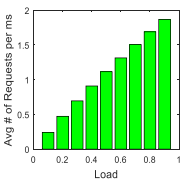


Fig. 14. Average # of req. received by MAN

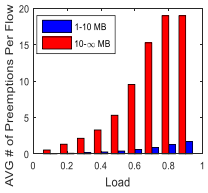
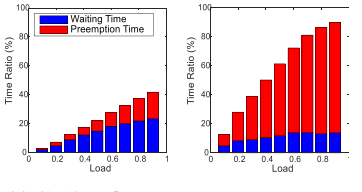


Fig. 15. Avg. # of preemptions per flow for 2 ranges of flow sizes



(a) (1-10MB] range (b) (10MB-∞) range

Fig. 16. Average portion of waiting and preemption times that a flow experiences to the total flow's completion time.

perform well. However, for Heavy workload including more big flows, the performance of schemes drops dramatically. Here, scheduling issue and handling big flows dominate, and the scheme which manages these issues better than others will achieve higher performance. That's why compared to other schemes, HyLine works very well under Heavy workload.

E. HyLine Deep Dive

In this section, a series of targeted simulations are conducted to dig deeper into HyLine's design.

Sensitivity to Threshold: To check our analysis in the §4.5, we use search workload and change the threshold identifying the two classes of traffic, and check the AFCT as the performance metric. Fig. 12 presents overall results in 60% load. Here, the results fit very well with our lower bound and upper bound analysis (Fig. 3). As we expected, for the thresholds below the lower bound, cost of doing centralized scheduling dominates, and for the ones above the upper bound, benefits of using centralized scheduler is not so much. So, in both cases, overall performance drops.

PFC: PFC, if used in a normal network, could cause the head-of-line blocking issue. However, since HyLine controls all of the 2nd class flows in the network, it prevents the head-of-line blocking issue for this class of flows. Moreover, PFC is used to prevent any drop of the 2nd class packets due to the increase in the number of 1st class flows at high load situations. To show the impact of using PFC at high loads, we use web search workload and do simulations with and without PFC feature in switches. Fig. 13 illustrates the improvement of the overall performance for the 2nd class of flows when PFC is turned on. In fact, PFC improves AFCT and 99th FCT by up to 13% and 15% respectively at high loads.

MAN: Here, we report MAN's performance measurements including average number of requests that MAN receives (Fig. 14), average waiting time (the time from when a flow first arrives at the end-host to when it receives first CTS (GO signal) from MAN), average preemption time (the total time that a flow is in STOP state (i.e., preempted by MAN)), and average number of preemptions that a flow experiences under web search workload. When load increases, as expected, number of preemptions per flow for the biggest flows (in (10MB-∞) range) increases (Fig. 15). However, since smaller flows (in (1-10MB] range) could be finished faster due to no competing bigger flows which are already stopped by MAN at the edge of network, the probability of being preempted during their transmission will be small. This is shown in Fig. 15.

Fig. 16 shows ratio of waiting and preemption times that on average a flow experiences to its total completion time across different loads for 2 different ranges of flow sizes. As mentioned earlier preemption time of flows in (1-10MB] range is small. Also, as loads increases waiting time of flows in this range slightly increases. The reason is that the flows which already

have got permission from MAN most likely have smaller remaining sizes compared to the new incoming flows, so new incoming flows will wait for the completion of these flows.

VI. Discussion

Flow Information: Previous studies show that for many DCN applications (e.g. web search, Hadoop [25], data processing), size of the flows are known at initiation time (For example, see §2.1 of [23]), and can be conveyed to lower layer (e.g., through a socket option). In other cases, when sizes of flows are not known precisely in advance, offline measurements enable applications to have an approximation of the flow sizes and use them later at run time. However, it is important to mention that based on DCN's traffic characteristics, HyLine does not require exact size information for 1st class flows (most of the DCN's flows), because it let them come to the network without scheduling them one by one, while all other size-aware schemes (e.g., [1], [2], [3]) need to know the exact size of all of the flows. Therefore, using HyLine, for most of the DCN's flows, these offline measurements will be just to check whether a flow is less than a threshold (e.g., 1MB).

Stopping vs. Terminating an Application: Although results shown in Fig. 16 indicate that most of the 2nd class flows have a very small preemption time, it is worth mentioning that stopping a 2nd class flow momentarily is not equal to terminating it. From the applications' point of view, it is more like TCP being in slow phase, so when a flow is stopped momentarily by MAN, connections are still there and applications are not terminated. Also, as mentioned before, HyLine modifies TCP to avoid having time-outs in STOP state and reacting to them as indication of packet loss.

Line Rate Transmission: With today's advances in both software (e.g., Intel DPDK [26], SR-IOV [27]) and hardware (e.g., [29], [30]), end-hosts can achieve line rate transmissions. However, if applications become the bottleneck of sending at line rate, for the 2nd class flows, they could simply add their maximum capable sending rate (maxRate) as part of their request message to MAN. MAN could consider those flows as flows generated by end-hosts having virtual maxRate-links (instead of their physical speed links). Therefore, without changing the logic, it could allow more flows to come and use same links.

VII. CONCLUSION

We presented HyLine a simple and practical flow scheduling design for DCNs. HyLine's path-aware scheduling policy exploiting the multipath nature of today's DCNs shows that load-balancing and flow scheduling design blocks are dependent blocks, and they should be designed together to minimize AFCT in DCNs. Moreover, HyLine's hybrid approach indicates that to reach high performance and minimize AFCT, it is unnecessary to use fine-grained scheduling structures trying to schedule every flow in DCNs by calculating and assigning either precise rates or priorities to them. In sum, Despite the simple nature of HyLine's design and the fact that it does not require any changes in the fabric, our evaluation results show that it outperforms recent flow scheduling solutions. That's why HyLine is a good candidate to be used in today's commodity DCNs, and why we believe that *performance meets simplicity at HyLine*.

REFERENCES

[1] M. Alizadeh et al., pFabric: Minimal near-optimal datacenter transport. In Proc. of SIGCOMM'13.

[2] C.-Y. Hong et al., Finishing Flows Quickly with Preemptive Scheduling. In Proc. of SIGCOMM, 2012.

[3] A. Munir et al., Friends, not Foes - Synthesizing Existing Transport Strategies for Data Center Networks. In Proc. of SIGCOMM 2014.

[4] W. Bai et al., Information-Agnostic Flow Scheduling for Commodity Data Centers. In NSDI, 2015.

[5] M. P. Grosvenor et al., Queues Don't Matter When You Can JUMP Them! In NSDI, 2015.

[6] F. R. Dogar et al., Decentralized Task-aware Scheduling for Data Center Networks. In Proc. of SIGCOMM'14.

[7] A. Greenberg et al., VL2: a scalable and flexible data center network. In Proc. of SIGCOMM, 2009.

[8] J. Perry et al., Fastpass: A Centralized "Zero-Queue" Datacenter Network. In Proc. of SIGCOMM, 2014.

[9] M. Alizadeh et al., Data center TCP (DCTCP). In Proc. of SIGCOMM, 2010.

[10] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[11] P. X. Gao et al., pHost: Distributed Near-optimal Datacenter Transport Over Commodity Network Fabric. In Proc. ACM CoNEXT, Dec. 2015.

[12] A. Dixit et al., On the Impact of Packet Spraying in Data Center Networks. In Proc. of INFOCOM, 2013.

[13] A. Singh et al., Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In Proc. of SIGCOMM, 2015.

[14] D. Zats, et al. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In Proc. of SIGCOMM, 2012.

[15] M. Al-Fares et al., Hedera: dynamic flow scheduling for data center networks. In Proc. of NSDI, 2010.

[16] M. Alizadeh et al., CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In Proc. of SIGCOMM, 2014.

[17] K. He et al., Presto: Edge-based Load Balancing for Fast Datacenter Networks. In Proc. of SIGCOMM, 2015.

[18] M. Al-Fares et al., A scalable, commodity data center network architecture. In Proc. of SIGCOMM, 2008.

[19] IEEE 802.1: 802.1Qbb - Priority-based Flow Control. http://www.ieee802.org/1/pages/802.1bb.html.

[20] HP FlexFabric 5700 Switch Series. http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04347352

[21] L.E. Schrage and L.W. Miller. The queue M/G/1 with the shortest processing remaining time discipline. Operations Research, 1966.

[22] A. Roy et al., Inside the social network's (datacenter) network. In Proc. of SIGCOMM, 2015.

[23] C. Wilson Better never than late: meeting deadlines in datacenter networks. In Proc. of SIGCOMM, 2011.

[24] Y. Lu et al., One More Queue is Enough: Minimizing Flow Completion Time with Explicit Priority Notification, In Proc. of INFOCOM, 2017.

[25] Hadoop. http://hadoop.apache.org/.

[26] DPDK. http://dpdk.org/.

[27] SR-IOV. https://www.pcisig.com/specifications/iov/.

[28] S. Abbasloo et al., "Cellular controlled delay TCP (C2TCP)," in IFIP Networking Conference (IFIP Networking) and Workshops, 2018. IEEE, 2018.

[29] P. K. Gupta, Intel® Xeon® + FPGA Platform for the Data Center. In FPL Workshop on Reconfigurable Computing for the Masses, Really?, 2015.

[30] SmartNIC. http://rnet-tech.com/smartnic.html

Dynamic Load Balancing with Tokens

Céline Comte
Nokia Bell Labs and Télécom ParisTech, University Paris-Saclay, France
celine.comte@nokia.com

***Abstract*—Efficiently exploiting the resources of data centers is a complex task that requires efficient and reliable load balancing and resource allocation algorithms. The former are in charge of assigning jobs to servers upon their arrival in the system, while the latter are responsible for sharing server resources between their assigned jobs. These algorithms should take account of various constraints, such as data locality, that restrict the feasible job assignments. In this paper, we propose a token-based mechanism that efficiently balances load between servers without requiring any knowledge on job arrival rates and server capacities. Assuming a balanced fair sharing of the server resources, we show that the resulting dynamic load balancing is insensitive to the job size distribution. Its performance is compared to that obtained under the best static load balancing and in an ideal system that would constantly optimize the resource utilization.**

I. Introduction

The success of cloud services encourages operators to scale out their data centers and optimize the resource utilization. The current trend consists in virtualizing applications instead of running them on dedicated physical resources [1]. Each server may then process several applications in parallel and each application may be distributed among several servers. Better understanding the dynamics of such server pools is a prerequisite for developing load balancing and resource allocation policies that fully exploit this new degree of flexibility.

Some recent works have tackled this problem from the point of view of queueing theory [2]–[5]. Their common feature is the adoption of a bipartite graph that translates practical constraints such as data locality into compatibility relations between jobs and servers. These models apply in various systems such as computer clusters, where the shared resource is the CPU [4], [5], and content delivery networks, where the shared resource is the server upload bandwidth [3]. However, these pool models do not consider simultaneously the impact of complex load balancing and resource allocation policies. The model of [2] lays emphasis on dynamic load balancing, assuming neither server multitasking nor job parallelism. The bipartite graph describes the initial compatibilities of incoming jobs, each of them being eventually assigned to a single server. On the other hand, [3]–[5] focus on the problem of resource allocation, assuming a static load balancing that assigns incoming jobs to classes at random, independently of the system state. The class of a job in the system identifies the set of servers that can be pooled to process it in parallel. The corresponding bipartite graph, connecting classes to servers, restricts the set of feasible resource allocations.

ISBN 978-3-903176-08-9

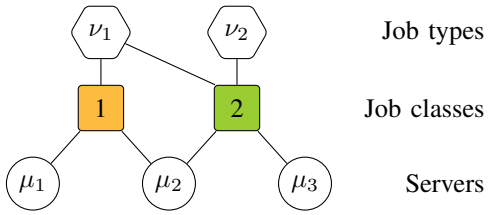


Fig. 1. A compatibility graph between types, classes and servers. Two consecutive servers can be pooled to process jobs in parallel. Thus there are two classes, one for servers 1 and 2 and another for servers 2 and 3. Type-1 jobs can be assigned to any class, while type-2 jobs can only be assigned to the latter. This restriction may result from data locality constraints for instance.

In this paper, we introduce a tripartite graph that explicitly differentiates the compatibilities of an incoming job from its actual assignment by the load balancer. This new model allows us to study the joint effect of load balancing and resource allocation. A toy example is shown in Fig. 1. Each incoming job has a type that defines its compatibilities; these may reflect its parallelization degree or locality constraints, for instance. Depending on the system state, the load balancer matches the job with a compatible class that subsequently determines its assigned servers. The upper part of our graph, which puts constraints on load balancing, corresponds to the bipartite graph of [2]; the lower part, which restricts the resource allocation, corresponds to the bipartite graph of [3]–[5].

We use this new framework to study load balancing and resource allocation policies that are *insensitive*, in the sense that they make the system performance independent of fine-grained traffic characteristics. This property is highly desirable as it allows service providers to dimension their infrastructure based on average traffic predictions only. It has been extensively studied in the queueing literature [3], [6]–[8]. In particular, insensitive load balancing policies were introduced in [8] in a generic queueing model, assuming an arbitrary insensitive allocation of the resources. These load balancing policies were defined as a generalization of the static load balancing described above, where the assignment probabilities of jobs to classes depend on both the job type and the system state, and are chosen to preserve insensitivity.

Our main contribution is an algorithm based on tokens that enforces such an insensitive load balancing without performing randomized assignments. More precisely, this is a *deterministic* implementation of an insensitive load balancing that adapts dynamically to the system state, under an arbitrary compatibility graph. The principle is as follows. The assignments are regulated through a bucket containing a fixed

number of tokens of each class. An incoming job seizes the longest available token among those that identify a compatible class, and is blocked if it does not find any. The rationale behind this algorithm is to use the release order of tokens as an information on the relative load of their servers: a token that has been available for a long time without being seized is likely to identify a server set that is less loaded than others. As we will see, our algorithm mirrors the first-come, first-served (FCFS) service discipline proposed in [5] to implement balanced fairness, which was defined in [7] as the most efficient insensitive resource allocation.

The closest existing algorithm we know is *assign longest idle server* (ALIS), introduced in reference [2] cited above. This work focuses on server pools without job parallel processing nor server multitasking. Hence, ALIS can be seen as a special case of our algorithm where each class identifies a server with a single token. The algorithm we propose is also related to the blocking version of Join-Idle-Queue [9] studied in [10]. More precisely, we could easily generalize our algorithm to server pools with several load balancers, each with their own bucket. The corresponding queueing model, still tractable using known results on networks of quasi-reversible queues [11], extends that of [10].

Organization of the paper: Section II recalls known facts about resource allocation in server pools. We describe a standard pool model based on a bipartite compatibility graph and explain how to apply balanced fairness in this model. Section III contains our main contributions. We describe our pool model based on a tripartite graph and introduce a new token-based insensitive load balancing mechanism. Numerical results are presented in Section IV.

II. Resource allocation

We first recall the model considered in [3]–[5] to study the problem of resource allocation in server pools. This model will be extended in Section III to integrate dynamic load balancing.

A. Model

We consider a pool of S servers. There are N job classes and we let $\mathcal{I} = \{1, \dots, N\}$ denote the set of class indices. For now, each incoming job is assigned to a compatible class at random, independently of the system state. For each $i \in \mathcal{I}$, the resulting arrival process of jobs assigned to class i is assumed to be Poisson with a rate $\lambda_i > 0$ that may depend on the job arrival rates, compatibilities and assignment probabilities. The number of jobs of class i in the system is limited by ℓ_i, for each $i \in \mathcal{I}$, so that a new job is blocked if its assigned class is already full. Job sizes are independent and exponentially distributed with unit mean. Each job leaves the system immediately after service completion.

The class of a job defines the set of servers that can be pooled to process it. Specifically, for each $i \in \mathcal{I}$, a job of class i can be served in parallel by any subset of servers within the non-empty set $\mathcal{S}_i \subset \{1, \dots, S\}$. This defines a bipartite compatibility graph between classes and servers, where there is an edge between a class and a server if the jobs of this class can be processed by this server. Fig. 2 shows a toy example.

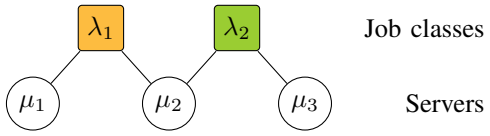


Fig. 2. A compatibility graph between classes and servers. Servers 1 and 3 are dedicated, while server 2 can serve both classes. The server sets associated with classes 1 and 2 are $\mathcal{S}_1 = \{1, 2\}$ and $\mathcal{S}_2 = \{2, 3\}$, respectively.

When a job is in service on several servers, its service rate is the sum of the rates allocated by each server to this job. For each $s = 1, \dots, S$, the capacity of server s is denoted by $\mu_s > 0$. We can then define a function μ on the power set of $\mathcal{I}$ as follows: for each $\mathcal{A} \subset \mathcal{I}$,

$$\mu(\mathcal{A}) = \sum_{s \in \bigcup_{i \in \mathcal{A}} \mathcal{S}_i} \mu_s$$

denotes the aggregate capacity of the servers that can process at least one class in $\mathcal{A}$, i.e., the maximum rate at which jobs of these classes can be served. μ is a submodular, non-decreasing set function [12]. It is said to be normalized because $\mu(\emptyset) = 0$.

B. Balanced fairness

We first recall the definition of balanced fairness [7], which was initially applied to server pools in [3]. Like processor sharing (PS) policy, balanced fairness assumes that the capacity of each server can be divided continuously between its jobs. It is further assumed that the resource allocation only depends on the number of jobs of each class in the system; in particular, all jobs of the same class receive service at the same rate.

The system state is described by the vector $x = (x_i : i \in \mathcal{I})$ of numbers of jobs of each class in the system. The state space is $\mathcal{X} = \{x \in \mathbb{N}^N : x \leq \ell\}$, where $\ell = (\ell_i : i \in \mathcal{I})$ is the vector of per-class constraints and the comparison $\leq$ is taken componentwise. For each $i \in \mathcal{I}$, we let $\phi_i(x)$ denote the total service rate allocated to class-i jobs in state x. It is assumed to be nonzero if and only if $x_i > 0$, in which case each job of class i receives service at rate $\frac{\phi_i(x)}{x_i}$.

Queueing model: Since all jobs of the same class receive service at the same rate, we can describe the evolution of the system with a network of N PS queues with state-dependent service capacities. For each $i \in \mathcal{I}$, queue i contains jobs of class i; the arrival rate at this queue is λ_i and its service capacity is $\phi_i(x)$ when the network state is x. An example is shown in Fig. 3 for the configuration of Fig. 2.

Capacity set: The compatibilities between classes and servers restrict the set of feasible resource allocations. Specifically, the vector $(\phi_i(x) : i \in \mathcal{I})$ of per-class service rates belongs to the following capacity set in any state $x \in \mathcal{X}$:

$$\Sigma = \left\{ \phi \in \mathbb{R}_+^N : \sum_{i \in \mathcal{A}} \phi_i \leq \mu(\mathcal{A}),\ \forall \mathcal{A} \subset \mathcal{I} \right\}.$$

As observed in [3], the properties satisfied by μ guarantee that Σ is a polymatroid [12].

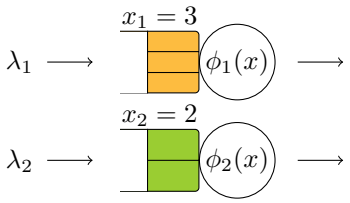


Fig. 3. An open Whittle network of $N = 2$ queues associated with the server pool of Fig. 2.

Balance function: It was shown in [6] that the resource allocation is insensitive if and only if there is a balance function Φ defined on $\mathcal{X}$ such that $\Phi(0) = 1$ and

$$\phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)}, \quad \forall x \in \mathcal{X}, \quad \forall i \in \mathcal{I}(x), \tag{1}$$

where e_i is the N-dimensional vector with 1 in component i and 0 elsewhere and $\mathcal{I}(x) = \{i \in \mathcal{I} : x_i > 0\}$ is the set of active classes in state x. Under this condition, the network of PS queues defined above is a Whittle network [13]. The insensitive resource allocations that respect the capacity constraints of the system are characterized by a balance function Φ such that, for all $x \in \mathcal{X} \setminus \{0\}$,

$$\Phi(x) \geq \frac{1}{\mu(\mathcal{A})} \sum_{i \in \mathcal{A}} \Phi(x - e_i), \quad \forall \mathcal{A} \subset \mathcal{I}(x), \ \mathcal{A} \neq \emptyset.$$

Recursively maximizing the overall service rate in the system is then equivalent to minimizing Φ by choosing

$$\Phi(x) = \max_{\substack{\mathcal{A} \subset \mathcal{I}(x), \\ \mathcal{A} \neq \emptyset}} \left(\frac{1}{\mu(\mathcal{A})} \sum_{i \in \mathcal{A}} \Phi(x - e_i) \right), \quad \forall x \in \mathcal{X} \setminus \{0\}.$$

The resource allocation defined by this balance function is called balanced fairness.

It was shown in [3] that balanced fairness is Pareto-efficient in polymatroid capacity sets, meaning that the total service rate $\sum_{i \in \mathcal{I}(x)} \phi_i(x)$ is always equal to the aggregate capacity $\mu(\mathcal{I}(x))$ of the servers that can process at least one active class. By (1), this is equivalent to

$$\Phi(x) = \frac{1}{\mu(\mathcal{I}(x))} \sum_{i \in \mathcal{I}(x)} \Phi(x - e_i), \quad \forall x \in \mathcal{X} \setminus \{0\}. \tag{2}$$

Stationary distribution: The Markov process defined by the system state x is reversible, with stationary distribution

$$\pi(x) = \pi(0)\Phi(x) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i}, \quad \forall x \in \mathcal{X}. \tag{3}$$

By insensitivity, the system state has the same stationary distribution if the jobs sizes within each class are only i.i.d., as long as the traffic intensity of class i (defined as the average quantity of work brought by jobs of this class per unit of time) is λ_i, for each $i \in \mathcal{I}$. A proof of this result is given in [6] for Cox distributions, which form a dense subset within the set of distributions of nonnegative random variables.

C. Job scheduling

We now describe the sequential implementation of balanced fairness that was proposed in [5]. This will lay the foundations for the results of Section III.

We still assume that a job can be distributed among several servers, but we relax the assumption that servers can process several jobs at the same time. Instead, each server processes its jobs sequentially in FCFS order. When a job arrives, it enters in service on every idle server within its assignment, if any, so that its service rate is the sum of the capacities of these servers. When the service of a job is complete, it leaves the system immediately and its servers are reallocated to the first job they can serve in the queue. Note that this sequential implementation also makes sense in a model where jobs are *replicated* over several servers instead of being processed *in parallel*. For more details, we refer the reader to [4] where the model with redundant requests was introduced.

Since the arrival order of jobs impacts the rate allocation, we need to detail the system state. We consider the sequence $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$, where n is the number of jobs in the system and c_p is the class of the p-th oldest job, for each $p = 1, \ldots, n$. $\emptyset$ denotes the empty state, with $n = 0$. The vector of numbers of jobs of each class in the system, corresponding to the state introduced in §II-B, is denoted by $|c| = (|c|_i : i \in \mathcal{I}) \in \mathcal{X}$. It does not define a Markov process in general. We let $\mathcal{I}(c) = \mathcal{I}(|c|)$ denote the set of active classes in state c. The state space of this detailed system state is $\mathcal{C} = \{c \in \mathcal{I}^* : |c| \leq \ell\}$.

Queueing model: Each job is in service on all the servers that were assigned this job but not those that arrived earlier. For each $p = 1, \ldots, n$, the service rate of the job in position p is thus given by

$$\sum_{s \in \mathcal{S}_{c_p} \setminus \bigcup_{q=1}^{p-1} \mathcal{S}_{c_q}} \mu_s = \mu(\mathcal{I}(c_1, \ldots, c_p)) - \mu(\mathcal{I}(c_1, \ldots, c_{p-1})),$$

with the convention that $(c_1, \ldots, c_{p-1}) = \emptyset$ if $p = 1$. The service rate of a job is independent of the jobs arrived later in the system. Additionally, the total service rate $\mu(\mathcal{I}(c))$ is independent of the arrival order of jobs. The corresponding queueing model is an order-independent (OI) queue [14], [15]. An example is shown in Fig. 4 for the configuration of Fig. 2.

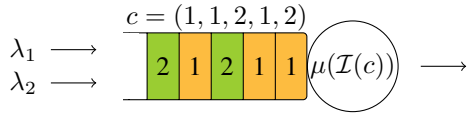


Fig. 4. An OI queue with $N = 2$ job classes associated with the server pool of Fig. 2. The job of class 1 at the head of the queue is in service on servers 1 and 2. The third job, of class 2, is in service on server 3. Aggregating the state c yields the state x of the Whittle network of Fig. 3.

Stationary distribution: The Markov process defined by the system state c is irreducible. The results of [15] show that this process is quasi-reversible, with stationary distribution

$$\pi(c) = \pi(\emptyset)\Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i}, \quad \forall c \in \mathcal{C}, \tag{4}$$

where Φ is defined recursively on $\mathcal{C}$ by $\Phi(\emptyset) = 1$ and

$$\Phi(c) = \frac{1}{\mu(\mathcal{I}(c))}\Phi(c_1, \ldots, c_{n-1}), \quad \forall c \in \mathcal{C} \setminus \{\emptyset\}. \quad (5)$$

We now go back to the aggregate state x giving the number of jobs of each class in the system. With a slight abuse of notation, we let

$$\pi(x) = \sum_{c:|c|=x} \pi(c) \quad \text{and} \quad \Phi(x) = \sum_{c:|c|=x} \Phi(c), \quad \forall x \in \mathcal{X}.$$

As observed in [5], [15], if follows from (4) that

$$\pi(x) = \pi(\emptyset) \left(\sum_{c:|c|=x} \Phi(c) \right) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i} = \pi(0)\Phi(x) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i}$$

in any state x. Using (5), we can show that Φ satisfies (2) with the initial condition $\Phi(0) = \Phi(\emptyset) = 1$. Hence, the stationary distribution of the aggregate system state x is exactly that obtained in §II-B under balanced fairness.

It was also shown in [5] that the average per-class resource allocation resulting from FCFS service discipline is balanced fairness. In other words, we have

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)} \mu_i(c), \quad \forall x \in \mathcal{X}, \quad \forall i \in \mathcal{I}(x),$$

where $\phi_i(x)$ is the total service rate allocated to class-i jobs in state x under balanced fairness, given by (1), and $\mu_i(c)$ denotes the service rate received by the first job of class i in state c under FCFS service discipline:

$$\mu_i(c) = \sum_{\substack{p=1 \\ c_p=i}}^{n} (\mu(\mathcal{I}(c_1, \ldots, c_p)) - \mu(\mathcal{I}(c_1, \ldots, c_{p-1}))).$$

Observe that, by (3) and (4), the rate equality simplifies to

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\Phi(c)}{\Phi(x)} \mu_i(c), \quad \forall x \in \mathcal{X}, \quad \forall i \in \mathcal{I}(x). \quad (6)$$

We will use this last equality later.

As it is, the FCFS service discipline is very sensitive to the job size distribution. [5] mitigates this sensitivity by frequently interrupting jobs and moving them to the end of the queue, in the same way as round-robin scheduling algorithm in the single-server case. In the queueing model, these interruptions and resumptions are represented approximately by random routing, which leaves the stationary distribution unchanged by quasi-reversibility [11], [13]. If the interruptions are frequent enough, then all jobs of a class tend to receive the same service rate on average, which is that obtained under balanced fairness. In particular, performance becomes approximately insensitive to the job size distribution within each class.

III. Load balancing

The previous section has considered the problem of resource sharing. We now focus on dynamic load balancing, using the fact that each job may be *a priori* compatible with several classes and assigned to one of them upon arrival. We first extend the model of §II-A to add this new degree of flexibility.

A. Model

We again consider a pool of S servers. There are N job classes and we let $\mathcal{I} = \{1, \ldots, N\}$ denote the set of class indices. The compatibilities between job classes and servers are described by a bipartite graph, as explained in §II-A. Additionally, we assume that the arrivals are divided into K types, so that the jobs of each type enter the system according to an independent Poisson process. Job sizes are independent and exponentially distributed with unit mean. Each job leaves the system immediately after service completion.

The type of a job defines the set of classes it can be assigned to. This assignment is performed instantaneously upon the job arrival, according to some decision rule that will be detailed later. For each $i \in \mathcal{I}$, we let $\mathcal{K}_i \subset \{1, \ldots, K\}$ denote the non-empty set of job types that can be assigned to class i. This defines a bipartite compatibility graph between types and classes, where there is an edge between a type and a class if the jobs of this type can be assigned to this class. Overall, the compatibilities are described by a tripartite graph between types, classes, and servers. Fig. 1 shows a toy example.

For each $k = 1, \ldots, K$, the arrival rate of type-k jobs in the system is denoted by $\nu_k > 0$. We can then define a function ν on the power set of $\mathcal{I}$ as follows: for each $\mathcal{A} \subset \mathcal{I}$,

$$\nu(\mathcal{A}) = \sum_{k \in \bigcup_{i \in \mathcal{A}} \mathcal{K}_i} \nu_k$$

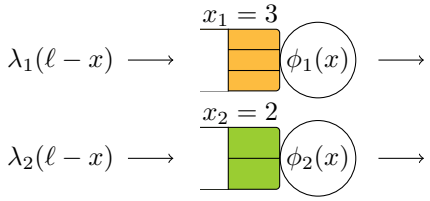
denotes the aggregate arrival rate of the types that can be assigned to at least one class in $\mathcal{A}$. ν satisfies the submodularity, monotonicity and normalization properties satisfied by the function μ of §II-A.

B. Randomized load balancing

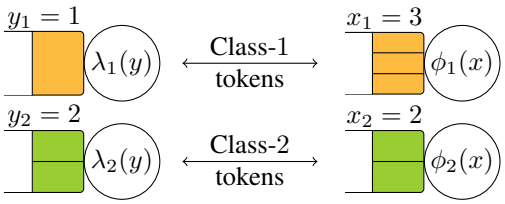
We now express the insensitive load balancing of [8] in our new server pool model. This extends the static load balancing considered earlier. Incoming jobs are assigned to classes at random, and the assignment probabilities depend not only on the job type but also on the system state. As in §II-B, we assume that the capacity of each server can be divided continuously between its jobs. The resources are allocated by applying balanced fairness in the capacity set defined by the bipartite compatibility graph between job classes and servers.

Open queueing model: We first recall the queueing model considered in [8] to describe the randomized load balancing. As in §II-B, jobs are gathered by class in PS queues with state-dependent service capacities given by (1). Hence, the type of a job is forgotten once it is assigned to a class.

Similarly, we record the job arrivals depending on the class they are assigned to, regardless of their type before the assignment. The Poisson arrival assumption ensures that, given the system state, the time before the next arrival at each class is exponentially distributed and independent of the arrivals at other classes. The rates of these arrivals result from the load balancing. We write them as functions of the vector $y = \ell - x$ of numbers of available positions at each class. Specifically, $\lambda_i(y)$ denotes the arrival rate of jobs assigned to class i when there are y_j available positions in class j, for each $j \in \mathcal{I}$.



(a) An open Whittle network with state-dependent arrival rates.



(b) A closed queueing system consisting of two Whittle networks.

Fig. 5. Alternative representations of a Whittle network associated with the server pool of Fig. 1. At most $\ell_1 = \ell_2 = 4$ jobs can be assigned to each class.

The system can thus be modeled by a network of N PS queues with state-dependent arrival rates, as shown in Fig. 5a.

Closed queueing model: We introduce a second queueing model that describes the system dynamics differently. It will later simplify the study of the insensitive load balancing by drawing a parallel with the resource allocation of §II-B.

Our alternative model stems from the following observation: since we impose limits on the number of jobs of each class, we can indifferently assume that the arrivals are limited by the intermediary of buckets containing tokens. Specifically, for each $i \in \mathcal{I}$, the assignments to class i are controlled through a bucket filled with ℓ_i tokens. A job that is assigned to class i removes a token from this bucket and holds it until its service is complete. The assignments to a class are suspended when the bucket of this class is empty, and they are resumed when a token of this class is released.

Each token is either held by a job in service or waiting to be seized by an incoming job. We consider a closed queueing model that reflects this alternation: a first network of N queues contains tokens held by jobs in service, as before, and a second network of N queues contains available tokens. For each $i \in \mathcal{I}$, a token of class i alternates between the queues indexed by i in the two networks. This is illustrated in Fig. 5b.

The state of the network containing tokens held by jobs in service is x. The queues in this network apply PS service discipline and their service capacities are given by (1). The state of the network containing available tokens is $y = \ell - x$. For each $i \in \mathcal{I}$, the service of a token at queue i in this network is triggered by the arrival of a job assigned to class i. The service capacity of this queue is thus equal to $\lambda_i(y)$ in state y. Since all tokens of the same class are exchangeable, we can assume indifferently that we pick one of them at random, so that the service discipline of the queue is PS.

Capacity set: The compatibilities between job types and classes restrict the set of feasible load balancings. Specifically, the vector $(\lambda_i(y) : i \in \mathcal{I})$ of per-class arrival rates belongs to the following capacity set in any state $y \in \mathcal{X}$:

$$\Gamma = \left\{ \lambda \in \mathbb{R}_+^N : \sum_{i \in \mathcal{A}} \lambda_i \leq \nu(\mathcal{A}),\ \forall \mathcal{A} \subset \mathcal{I} \right\}.$$

The properties satisfied by ν guarantee that Γ is a polymatroid.

Balance function: Our token-based reformulation allows us to interpret dynamic load balancing as a problem of resource allocation in the network of queues containing available tokens. This will allow us to apply the results of §II-B.

It was shown in [8] that the load balancing is insensitive if and only if there is a balance function Λ defined on $\mathcal{X}$ such that $\Lambda(0) = 1$, and

$$\lambda_i(y) = \frac{\Lambda(y - e_i)}{\Lambda(y)}, \quad \forall y \in \mathcal{X}, \quad \forall i \in \mathcal{I}(y). \tag{7}$$

Under this condition, the network of PS queues containing available tokens is a Whittle network.

The Pareto-efficiency of balanced fairness in polymatroid capacity sets can be understood as follows in terms of load balancing. We consider the balance function Λ defined recursively on $\mathcal{X}$ by $\Lambda(0) = 1$ and

$$\Lambda(y) = \frac{1}{\nu(\mathcal{I}(y))} \sum_{i \in \mathcal{I}(y)} \Lambda(y - e_i), \quad \forall y \in \mathcal{X} \setminus \{0\}. \tag{8}$$

Then Λ defines a load balancing that belongs to the capacity set Γ in each state y. By (7), this load balancing satisfies

$$\sum_{i \in \mathcal{I}(y)} \lambda_i(y) = \nu(\mathcal{I}(y)), \quad \forall y \in \mathcal{X},$$

meaning that an incoming job is accepted whenever it is compatible with at least one available token.

Stationary distribution: The Markov process defined by the system state x is reversible, with stationary distribution

$$\pi(x) = \frac{1}{G} \Phi(x) \Lambda(\ell - x), \quad \forall x \in \mathcal{X}, \tag{9}$$

where G is a normalization constant. Note that we could symmetrically give the stationary distribution of the Markov process defined by the vector $y = \ell - x$ of numbers of available tokens. As mentioned earlier, the insensitivity of balanced fairness is preserved by the load balancing.

C. Deterministic token mechanism

Our closed queueing model reveals that the randomized load balancing is dual to the balanced fair resource allocation. This allows us to propose a new *deterministic* load balancing algorithm that mirrors the FCFS service discipline of §II-C. This algorithm can be combined indifferently with balanced fairness or with the sequential FCFS scheduling; in both cases, we show that it implements the load balancing defined by (7).

All available tokens are now sorted in order of release in a single bucket. The longest available tokens are in front. An incoming job scans the bucket from beginning to end and seizes the first compatible token; it is blocked if it does not find any. For now, we assume that the server resources are allocated to the accepted jobs by applying the FCFS service

discipline of §II-C. When the service of a job is complete, its token is released and added to the end of the bucket.

We describe the system state with a couple (c,t) retaining both the arrival order of jobs and the release order of tokens. Specifically, $c=(c_1,\dots,c_n)\in\mathcal{C}$ is the sequence of classes of (tokens held by) jobs in service, as before, and $t=(t_1,\dots,t_m)\in\mathcal{C}$ is the sequence of classes of available tokens, ordered by release, so that t_1 is the class of the longest available token. Given the total number of tokens of each class in the system, any feasible state satisfies $|c|+|t|=\ell$.

Queueing model: Depending on its position in the bucket, each available token is seized by any incoming job whose type is compatible with this token but not with the tokens released earlier. For each $p=1,\dots,m$, the token in position p is thus seized at rate

$$\sum_{k\in\mathcal{K}_{t_p}\setminus\bigcup_{q=1}^{p-1}\mathcal{K}_{t_q}}\nu_k=\nu(\mathcal{I}(t_1,\dots,t_p))-\nu(\mathcal{I}(t_1,\dots,t_{p-1})).$$

The seizing rate of a token is independent of the tokens released later. Additionally, the total rate at which available tokens are seized is $\nu(\mathcal{I}(y))$, independently of their release order. The bucket can thus be modeled by an OI queue, where the service of a token is triggered by the arrival of a job that seizes this token.

The evolution of the sequence of tokens held by jobs in service also defines an OI queue, with the same dynamics as in §II-C. Overall, the system can be modeled by a closed tandem network of two OI queues, as shown in Fig. 6.

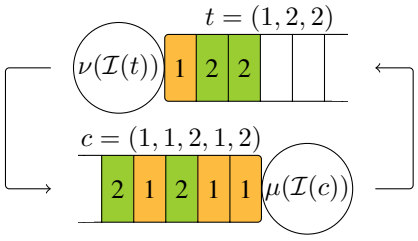


Fig. 6. A closed tandem network of two OI queues associated with the server pool of Fig. 1. At most $\ell_1=\ell_2=4$ jobs can be assigned to each class. The state is (c,t), with $c=(1,1,2,1,2)$ and $t=(1,2,2)$. The corresponding aggregate state is that of the network of Fig. 5. An incoming job of type 1 would seize the available token in first position (of class 1), while an incoming job of type 2 would seize the available token in second position (of class 2).

Stationary distribution: Assuming $\mathcal{S}_i\neq\mathcal{S}_j$ or $\mathcal{K}_i\neq\mathcal{K}_j$ for each pair $\{i,j\}\subset\mathcal{I}$ of classes, the Markov process defined by the detailed state (c,t) is irreducible. The proof, omitted for brevity, is given in the technical report [16]. Known results on networks of quasi-reversible queues [11] then show that this process is quasi-reversible, with stationary distribution

$$\pi(c,t)=\frac{1}{G}\Phi(c)\Lambda(t),\quad\forall c,t\in\mathcal{C}:|c|+|t|=\ell,$$

where Φ is defined by the recursion (5) and the initial step $\Phi(\emptyset)=1$, as in §II-C; similarly, Λ is defined recursively on $\mathcal{C}$ by $\Lambda(\emptyset)=1$ and

$$\Lambda(t)=\frac{1}{\nu(\mathcal{I}(t))}\Lambda(t_1,\dots,t_{m-1}),\quad\forall t\in\mathcal{C}\setminus\{\emptyset\}.$$

We go back to the aggregate state x giving the number of tokens of each class held by jobs in service. With a slight abuse of notation, we define its stationary distribution by

$$\pi(x)=\sum_{c:|c|=x}\sum_{t:|t|=\ell-x}\pi(c,t),\quad\forall x\in\mathcal{X}.\tag{10}$$

As in §II-C, we can show that we have

$$\pi(x)=\frac{1}{G}\Phi(x)\Lambda(\ell-x),\quad\forall x\in\mathcal{X},$$

where the functions Φ and Λ are defined on $\mathcal{X}$ by

$$\Phi(x)=\sum_{c:|c|=x}\Phi(c)\quad\text{and}\quad\Lambda(y)=\sum_{t:|t|=y}\Lambda(t),\quad\forall x,y\in\mathcal{X},$$

respectively. These functions Φ and Λ satisfy the recursions (2) and (8), respectively, with the initial conditions $\Phi(0)=\Lambda(0)=1$. Hence, the aggregate stationary distribution of the system state x is exactly that obtained in §III-B by combining the randomized load balancing with balanced fairness.

Also, using the definition of Λ, we can rewrite (6) as follows: for each $x\in\mathcal{X}$ and $i\in\mathcal{I}(x)$,

$$\begin{aligned}\phi_i(x)&=\sum_{c:|c|=x}\frac{\frac{1}{G}\Phi(c)\sum_{t:|t|=\ell-x}\Lambda(t)}{\frac{1}{G}\Phi(x)\Lambda(\ell-x)}\mu_i(c),\\&=\sum_{c:|c|=x}\sum_{t:|t|=\ell-x}\frac{\pi(c,t)}{\pi(x)}\mu_i(c).\end{aligned}$$

Hence, the average per-class service rates are still as defined by balanced fairness. By symmetry, it follows that the average per-class arrival rates, ignoring the release order of tokens, are as defined by the randomized load balancing. Specifically, for each $y\in\mathcal{X}$ and $i\in\mathcal{I}(y)$, we have

$$\lambda_i(y)=\sum_{c:|c|=\ell-y}\sum_{t:|t|=y}\frac{\pi(c,t)}{\pi(\ell-y)}\nu_i(t),$$

where $\lambda_i(y)$ is the arrival rate of jobs assigned to class i in state y under the randomized load balancing, given by (7), and $\nu_i(t)$ denotes the rate at which the first available token of class i is seized under the deterministic load balancing:

$$\nu_i(t)=\sum_{\substack{p=1\\t_p=i}}^{m}(\nu(\mathcal{I}(t_1,\dots,t_p))-\nu(\mathcal{I}(t_1,\dots,t_{p-1}))).$$

As in §II-C, the stationary distribution of the system state is unchanged by the addition of random routing, as long as the average traffic intensity of each class remains constant. Hence we can again reach some approximate insensitivity to the job size distribution within each class by enforcing frequent job interruptions and resumptions.

Application with balanced fairness: As announced earlier, we can also combine our token-based load balancing algorithm with balanced fairness. The assignment of jobs to classes is still regulated by a single bucket containing available tokens, sorted in release order, but the resources are now allocated according to balanced fairness. The corresponding queueing

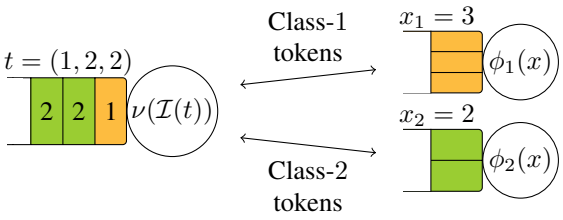


Fig. 7. A closed queueing system, consisting of an OI queue and a Whittle network, associated with the server pool of Fig. 1. At most $\ell_1 = \ell_2 = 4$ jobs can be assigned to each class.

model consists of an OI queue and a Whittle network, as represented in Fig. 7.

The intermediary state (x, t), retaining the release order of available tokens but not the arrival order of jobs, defines a Markov process. Its stationary distribution follows from known results on networks of quasi-reversible queues [11]:

$$\pi(x,t) = \frac{1}{G}\Phi(x)\Lambda(t), \quad \forall x \in \mathcal{X},\ \forall t \in \mathcal{C} :\ x + |t| = \ell.$$

We can show as before that the average per-class arrival rates, ignoring the release order of tokens, are as defined by the dynamic load balancing of §III-B.

The insensitivity of balanced fairness to the job size distribution within each class is again preserved. The proof of [6] for Cox distributions extends directly. Note that this does no imply that performance is insensitive to the job size distribution *within each type*. Indeed, if two job types with different size distributions can be assigned to the same class, then the distribution of the job sizes within this class may be correlated to the system state upon their arrival. This point will be assessed by simulation in Section IV.

Observe that our token-based mechanism can be applied to balance the load between the queues of an arbitrary Whittle network, as represented in Fig. 7, independently of the system considered. Examples or such systems are given in [8].

IV. Numerical results

We finally consider two examples that give insights on the performance of our token-based algorithm. We especially make a comparison with the static load balancing of Section II and assess the insensitivity to the job size distribution within each type. We refer the reader to [17] for a large-scale analysis in homogeneous pools with a single job type, along with a comparison with other (non-insensitive) standard policies.

Performance metrics for Poisson arrival processes and exponentially distributed sizes with unit mean follow from (9). By insensitivity, these also give the performance when job sizes within each class are i.i.d., as long as the traffic intensity is unchanged. We resort to simulations to evaluate performance when the job size distribution is type-dependent.

Performance is measured by the job blocking probability and the resource occupancy. For each $k = 1, \dots, K$, we let

$$\beta_k = \frac{1}{G} \sum_{\substack{x \le \ell: \\ x_i = \ell_i,\ \forall i \in \mathcal{I}: k \in \mathcal{K}_i}} \Phi(x)\Lambda(\ell - x)$$

denote the probability that a job of type k is blocked upon arrival. The equality follows from PASTA property [13]. Symmetrically, for each $s = 1, \dots, S$, we let

$$\psi_s = \frac{1}{G} \sum_{\substack{x \le \ell: \\ x_i = 0,\ \forall i \in \mathcal{I}: s \in \mathcal{S}_i}} \Phi(x)\Lambda(\ell - x)$$

denote the probability that server s is idle. These quantities are related by the conservation equation

$$\sum_{k=1}^{K} \nu_k (1 - \beta_k) = \sum_{s=1}^{S} \mu_s (1 - \psi_s). \tag{11}$$

We define respectively the average blocking probability and the average resource occupancy by

$$\beta = \frac{\sum_{k=1}^{K} \nu_k \beta_k}{\sum_{k=1}^{K} \nu_k} \quad \text{and} \quad \eta = \frac{\sum_{s=1}^{S} \mu_s (1 - \psi_s)}{\sum_{s=1}^{S} \mu_s}.$$

There is a simple relation between β and η. Indeed, if we let $\rho = (\sum_{k=1}^{K} \nu_k)/(\sum_{s=1}^{S} \mu_s)$ denote the total load in the system, then we can rewrite (11) as $\rho(1 - \beta) = \eta$.

As expected, minimizing the average blocking probability is equivalent to maximizing the average resource occupancy. It is however convenient to look at both metrics in parallel. As we will see, when the system is underloaded, jobs are almost never blocked and it is easier to describe the (almost linear) evolution of the resource occupancy. On the contrary, when the system is overloaded, resources tend to be maximally occupied and it is more interesting to focus on the blocking probability.

Observe that any stable server pool satisfies the conservation equation (11). In particular, the average blocking probability β in a stable system cannot be less than $1 - \frac{1}{\rho}$ when $\rho > 1$. A similar argument applied to each job type imposes that

$$\beta_k \ge \max\left(0, 1 - \frac{1}{\nu_k} \sum_{s \in \bigcup_{i: k \in \mathcal{K}_i} \mathcal{S}_i} \mu_s\right), \tag{12}$$

for each $k = 1, \dots, K$.

A. A single job type

We first consider a pool of $S = 10$ servers with a single type of jobs ($K = 1$), as shown in Fig. 8. Each class identifies a unique server and each job can be assigned to any class. Half of the servers have a unit capacity μ and the other half have capacity 4μ. Each server has $\ell = 6$ tokens and applies PS policy to its jobs. We do not look at the insensitivity to the job size distribution in this case, as there is a single job type.

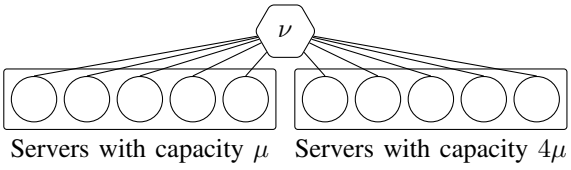


Fig. 8. A server pool with a single job type. Classes are omitted because each of them corresponds to a single server.

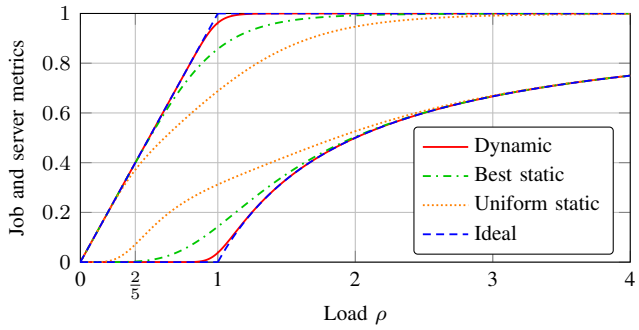


Fig. 9. Performance of the dynamic load balancing in the pool of Fig. 8. Average blocking probability (bottom plot) and resource occupancy (top plot).

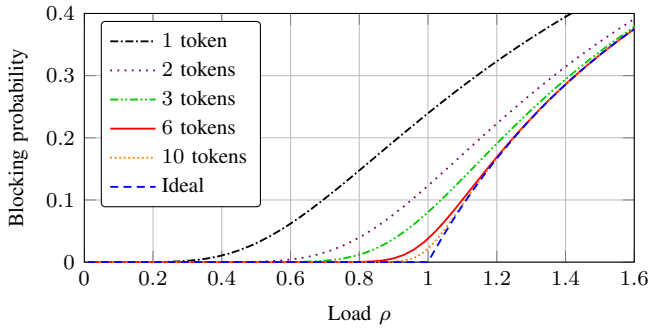


Fig. 10. Impact of the number of tokens on the average blocking probability under the dynamic load balancing in the pool of Fig. 8.

Comparison: We compare the performance of our algorithm with that of the static load balancing of Section II, where each job is assigned to a server at random, independently of system state, and blocked if its assigned server is already full. We consider two variants, *best static* and *uniform static*, where the assignment probabilities are proportional to the server capacities and uniform, respectively. *Ideal* refers to the lowest average blocking probability that complies with the system stability. According to (11), it is 0 when $\rho \leq 1$ and $1-\frac{1}{\rho}$ when $\rho > 1$. One can think of it as the performance in an ideal server pool where resources would be constantly optimally utilized. The results are shown in Fig. 9.

The performance gain of our algorithm compared to the static policies is maximal near the critical load $\rho = 1$, which is also the area where the delta with *ideal* is maximal. Elsewhere, all load balancing policies have a comparable performance. Our intuition is as follows: when the system is underloaded, servers are often available and the blocking probability is low anyway; when the system is overloaded, resources are congested and the blocking probability is high whichever scheme is utilized. Observe that the performance under *uniform static* deteriorates faster, even when $\rho < 1$, because the servers with the lowest capacity, concentrating half of the arrivals with only $\frac{1}{5}$-th of the service capacity, are congested whenever $\rho > \frac{2}{5}$. This stresses the need for accurate rate estimations under a static load balancing.

Asymptotics when the number of tokens increases: We now focus on the impact of the number of tokens on the performance of the dynamic load balancing. A direct calculation shows that the average blocking probability decreases with the number ℓ of tokens per server, and tends to *ideal* as $\ell \to +\infty$. Intuitively, having many tokens gives a long run feedback on the server loads without blocking arrivals more than necessary (to preserve stability). The results are shown in Fig. 10.

We observe that the convergence to the asymptotic ideal is quite fast. The largest gain is obtained with small values of ℓ and the performance is already close to the optimal with $\ell = 10$ tokens per server. Hence, we can reach a low blocking probability even when the number of tokens is limited, for instance to guarantee a minimum service rate per job or respect multitasking constraints on the servers.

B. Several job types

We now consider a pool of $S = 6$ servers, all with the same unit capacity μ, as shown in Fig. 11. As before, there is no parallel processing. Each class identifies a unique server that applies PS policy to its jobs and has $\ell = 6$ tokens. There are two job types with different arrival rates and compatibilities. Type-1 jobs have a unit arrival rate ν and can be assigned to any of the first four servers. Type-2 jobs arrive at rate 4ν and can be assigned to any of the last four servers. Thus only two servers can be accessed by both types. Note that heterogeneity now lies in the job arrival rates and not in the server capacities.

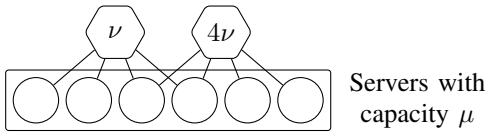


Fig. 11. A server pool with two job types.

Comparison: We again consider two variants of the static load balancing: *best static*, in which the assignment probabilities are chosen so as to homogenize the arrival rates at the servers as far as possible, and *uniform static*, in which the assignment probabilities are uniform. Note that *best static* assumes that the arrival rates of the job types are known, while *uniform static* does not. As before, *ideal* refers to the lowest average blocking probability that complies with the system stability. The results are shown in Fig. 12.

Regardless of the policy, the slope of the resource occupancy breaks down near the critical load $\rho = \frac{5}{6}$. The reason is that the last four servers support at least $\frac{4}{5}$-th of the arrivals with only $\frac{2}{3}$-rd of the service capacity, so that their effective load is $\frac{6}{5}\rho$. It follows from (12) that the average blocking probability in a stable system cannot be less than $\frac{4}{5}(1-\frac{5}{6}\frac{1}{\rho})$ when $\rho \geq \frac{5}{6}$. Under *ideal*, the slope of the resource occupancy breaks down again at $\rho = \frac{5}{3}$. This is the point where the first two servers cannot support the load of type-1 jobs by themselves anymore.

Otherwise, most of the observations of §IV-A are still valid. The performance gain of the dynamic load balancing compared to *best static* is maximal near the first critical load $\rho = \frac{5}{6}$. Its delta with *ideal* is maximal near $\rho = \frac{5}{6}$ and $\rho = \frac{5}{3}$.

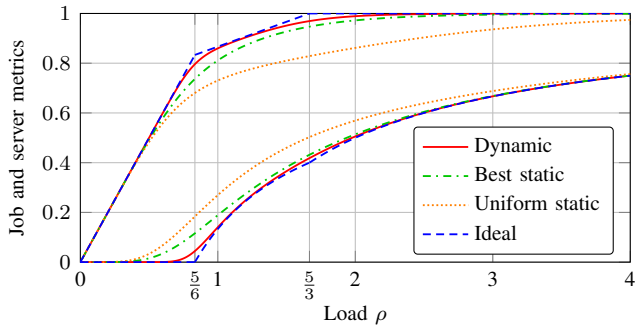


Fig. 12. Performance of the dynamic load balancing in the pool of Fig. 11. Average blocking probability (bottom plot) and resource occupancy (top plot).

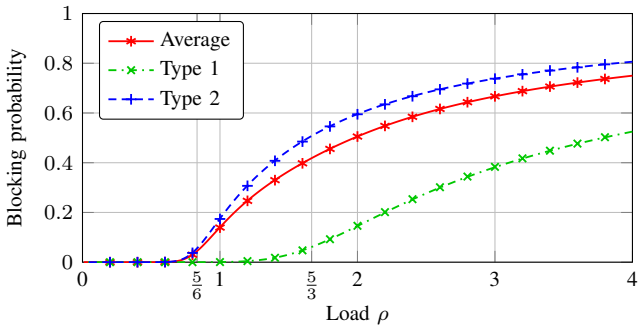


Fig. 13. Blocking probability under the dynamic load balancing in the server pool of Fig. 11, with either exponentially distributed job sizes (line plots) or hyperexponentially distributed sizes (marks). Each simulation point is the average of 100 independent runs, each built up of 10^6 jumps after a warm-up period of 10^6 jumps. The corresponding 95% confidence interval, not shown on the figure, does not exceed ± 0.001 around the point.

Elsewhere, all schemes have a similar performance, except for *uniform static* that deteriorates faster.

Overall, these numerical results show that our dynamic load balancing algorithm often outperforms *best static* and is close to *ideal*. The configurations (not shown here) where it was not the case involved very small pools, with job arrival rates and compatibilities opposite to the server capacities. Our intuition is that our algorithm performs better when the pool size or the number of tokens allow for some diversity in the assignments.

(In)sensitivity: We finally evaluate the sensitivity of our algorithm to the job size distribution within each type. Fig. 13 shows the results. Lines give the performance when job sizes are exponentially distributed with unit mean, as before. Marks, obtained by simulation, give the performance when the job size distribution within each type is hyperexponential: $\frac{1}{3}$-rd of type-1 jobs have an exponentially distributed size with mean 2 and the other $\frac{2}{3}$-rd have an exponentially distributed size with mean $\frac{1}{2}$; similarly, $\frac{1}{6}$-th of type-2 jobs have an exponentially distributed size with mean 5 and the other $\frac{5}{6}$-th have an exponentially distributed size with mean $\frac{1}{5}$.

The similarity of the exact and simulation results suggests that insensitivity is preserved even when the job size distribution is type-dependent. Further evaluations, involving other job size distributions, would be necessary to conclude.

Also observe that the blocking probability of type-1 jobs increases near the load $\rho = \frac{5}{3}$, which is twice less than the upper bound $\rho = \frac{10}{3}$ given by (12). This suggests that the dynamic load balancing compensates the overload of type-2 jobs by rejecting more jobs of type 1.

V. Conclusion

We have introduced a new server pool model that explicitly distinguishes the compatibilities of a job from its actual assignment by the load balancer. Expressing the results of [8] in this new model has allowed us to see the problem of load balancing in a new light. We have derived a deterministic, token-based implementation of a dynamic load balancing that preserves the insensitivity of balanced fairness to the job size distribution within each class. Numerical results have assessed the performance of this algorithm.

For the future works, we would like to evaluate the performance of our algorithm in broader classes of server pools. We are also interested in proving its insensitivity to the job size distribution within each type.

References

[1] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed. Morgan & Claypool Publishers, 2013.

[2] I. Adan and G. Weiss, "A loss system with skill-based servers under assign to longest idle server policy," *Probability in the Engineering and Informational Sciences*, vol. 26, no. 3, p. 307321, 2012.

[3] V. Shah and G. de Veciana, "High-Performance Centralized Content Delivery Infrastructure: Models and Asymptotics," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1674–1687, Oct. 2015.

[4] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 1, pp. 347–360, Jun. 2015.

[5] T. Bonald and C. Comte, "Balanced fair resource sharing in computer clusters," *Performance Evaluation*, vol. 116, no. Supplement C, pp. 70–83, Nov. 2017.

[6] T. Bonald and A. Proutière, "Insensitivity in processor-sharing networks," *Performance Evaluation*, vol. 49, no. 1, pp. 193–209, Sep. 2002.

[7] ——, "Insensitive bandwidth sharing in data networks," *Queueing Syst.*, vol. 44, no. 1, pp. 69–100, 2003.

[8] T. Bonald, M. Jonckheere, and A. Proutière, "Insensitive load balancing," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 367–377, Jun. 2004.

[9] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg, "Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, Nov. 2011.

[10] M. van der Boor, S. Borst, and J. van Leeuwaarden, "Load balancing in large-scale systems with multiple dispatchers," in *Proceedings of INFOCOM 2017*, 2017.

[11] F. P. Kelly, *Reversibility and Stochastic Networks*. New York, NY, USA: Cambridge University Press, 2011.

[12] S. Fujishige, *Submodular Functions and Optimization, Volume 58 - 2nd Edition*. Elsevier Science, Jul. 2005.

[13] R. Serfozo, *Introduction to Stochastic Networks*, ser. Stochastic Modelling and Applied Probability. Springer New York, 1999.

[14] S. A. Berezner and A. E. Krzesinski, "Order independent loss queues," *Queueing Systems*, vol. 23, no. 1-4, pp. 331–335, Mar. 1996.

[15] A. E. Krzesinski, "Order independent queues," in *Queueing Networks: A Fundamental Approach*, R. J. Boucherie and N. M. van Dijk, Eds. Boston, MA: Springer US, 2011, pp. 85–120.

[16] C. Comte, "Dynamic Load Balancing with Tokens," *arXiv:1804.01783 [cs]*, Apr. 2018, technical report, arXiv: 1804.01783. [Online]. Available: http://arxiv.org/abs/1804.01783

[17] M. Jonckheere and B. J. Prabhu, "Asymptotics of insensitive load balancing and blocking phases," *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 311–322, Jun. 2016.

Moving Bits with a Fleet of Shared Virtual Routers

Pradeeban Kathiravelu*†, Marco Chiesa‡, Pedro Marcos§, Marco Canini¶ and Luís Veiga*
*INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa †Université catholique de Louvain
‡KTH Royal Institute of Technology §UFRGS/FURG ¶KAUST

***Abstract*—The steady decline of IP transit prices in the past two decades has helped fuel the growth of traffic demands in the Internet ecosystem. Despite the declining unit pricing, bandwidth costs remain significant due to ever-increasing scale and reach of the Internet, combined with the price disparity between the Internet's core hubs versus remote regions. In the meantime, cloud providers have been auctioning underutilized computing resources in their marketplace as spot instances for a much lower price, compared to their on-demand instances. This state of affairs has led the networking community to devote extensive efforts to cloud-assisted networks — the idea of offloading network functionality to cloud platforms, ultimately leading to more flexible and highly composable network service chains.**

We initiate a critical discussion on the economic and technological aspects of leveraging cloud-assisted networks for Internet-scale interconnections and data transfers. Namely, we investigate the prospect of constructing a large-scale virtualized network provider that does not own any fixed or dedicated resources and runs atop several spot instances. We construct a cloud-assisted overlay as a virtual network provider, by leveraging third-party cloud spot instances. We identify three use case scenarios where such approach will not only be economically and technologically viable but also provide performance benefits compared to current commercial offerings of connectivity and transit providers.

I. Introduction

The massive amount of content shared on the Internet, along with the bandwidth requirements to provide higher Quality of Experience (QoE) for latency-sensitive applications such as online gaming and video conferencing, has resulted in ever-increasing bandwidth demand and an urgent desire for flexibility in interconnections by network operators.

Cloud-assisted networks have been recently proposed, to increase performance, flexibility, and reliability of wide area networks [31]. They leverage cloud resources to compose large-scale overlay networks. This approach is appealing because cloud platforms generally guarantee high levels of availability through various levels of Service Level Agreements (SLAs) [19]. Moreover, major cloud providers such as Amazon have built their own global backbone network [40]. Therefore, they do not rely on the transit providers for their data transfer. These cloud networks are well provisioned and maintained, which makes them better than the Internet paths regarding loss rate and jitter [32]. Based on these observations, companies such as Teridion [42] and Cloudflare [23] offer cloud-assisted networks for SaaS providers as a premium service for a higher price compared to using standard Internet-based connectivity.

However, the use of cloud-assisted overlays in a broader set of use cases, such as their use by end users or enterprises for regular data transfers and the economic viability of such cases, is not well studied. We argue that the feasibility of cloud-assisted overlay solutions deserves a broader study informed by three key factors that we identify as follows.

Geographical price disparity. First, the oligopoly of a limited number of connectivity providers and a substantial dependence on expensive long-haul links to the US or EU for international connectivity, have caused a higher price for IP transit in the remote Internet regions [20]. For instance, prices for 10 Gbps Ethernet (10 GbE) bandwidth remain up to 20 times more expensive in São Paulo and Sydney, compared to EU and USA. This disparity is even more pronounced in remote regions such as Central Asia and Sub-Saharan Africa. For example, as of 2014, while the transit cost per Mbps per month was 0.94$ in the US [37], it was 15$ in Kazakhstan and 347$ in Uzbekistan [35]. Even though the average IP transit prices at major Internet hubs have fallen by an annual 61% during the past two decades, this decline has been much less pronounced elsewhere [37]. Consequently, the economic viability and incentives of a cloud-assisted solution significantly depend on its geographical location.

On-demand bandwidth. Second, even though transit providers currently offer bandwidth with minimum commitment, as low as 10 Mbps [24], interconnection contracts shorter than one month are still uncommon. To rectify this limitation, companies such as Epsilon [4] and PacketFabric [8] are working towards making bandwidth a tradeable utility, steering up dynamic interconnections among their users. Furthermore, companies like Megaport [6] and Console Connect [3] provide scalable point-to-point connectivity to cloud and network providers. Aligning to these recent developments, the pay-per-use and per-second billing of cloud providers can enable cloud-assisted solutions to offer dynamic interconnections with no commitment to time or usage.

Low-cost cloud resources. Third, *spot markets* of cloud providers such as Amazon Web Services (AWS) and Google Cloud Platform (GCP) offer their spare compute instances, with the same resources and capabilities as their on-demand counterparts, at a much lower price. Nevertheless, spot instances can be suddenly interrupted with a notification period of up to two minutes. Applications that can tolerate the volatile nature of the spot instances can use them as an economical alternative to the on-demand ones. These spot markets, typically underutilized, have idle and affordable resources in multiple regions. The economic viability of cloud-assisted networks depends on the need for connectivity services that are more dynamic than the traditional ones. Resilient architectures built

ISBN 978-3-903176-08-9

atop spot instances can bring cloud expenditures down enough to make cloud-assisted overlays profitable.

Given the above premises, we set out to investigate the following research questions: i) Can a cloud-assisted network built on several spot instances be a viable solution to realize an on-demand virtual Network-as-a-Service provider, i.e., a network provider built over multiple cloud offerings and that does not own any fixed or dedicated resources? ii) If so, what are its possible usage scenarios and under what conditions would this approach be economically sustainable for a network provider? iii) When would this approach be cheaper than existing alternative connectivity providers, including transit providers, Internet Service Providers (ISPs), and Multiprotocol Label Switching (MPLS) network providers? iv) how stable are today's spot instances? v) If not competitive on price, would this approach be able to provide higher performance and/or additional features than the alternatives?

We study the technological and economic viability of such a cloud-assisted network and propose $NetUber$ as an efficient architecture to realize it. $NetUber$ consists of a broker that i) purchases spot VMs from the cloud providers, and ii) creates an overlay network over the multiple spot VMs to function as a large-scale inter-region connectivity provider to the customers who would buy connectivity directly from $NetUber$. Thus, $NetUber$ operates as an on-demand virtual connectivity provider running atop virtual routers in the spot VMs. By leveraging the memory and CPU of the acquired spot instances, we further envision a deployment of auxiliary services such as compression-as-a-service [1] and encryption-as-a-service [2], offering an optional compressed or encrypted data transfer between the regions.

Our primary contributions are: i) an economic model to exploit spot markets for direct secured connectivity between pairs of endpoints, and ii) an inter-cloud approach that leverages spot VMs in building a reliable virtual connectivity provider. When compared to traditional flat-price connectivity providers, our extensive evaluation shows that i) $NetUber$ best suits the needs of small dynamic monthly transfers up to at least 50 TB and ii) $NetUber$ cuts Internet latencies up to a factor of 30%. We see our contributions as a first step towards a more systematic understanding of the next-generation overlay interconnection networks.

We discuss the current potential for cloud-assisted networks in Section II. We then elaborate the potential deployments for the identified use cases of $NetUber$ in Section III. We illustrate the opportunities for $NetUber$ through an economic analysis in Section IV. We evaluate $NetUber$ against the current offerings in Section V. We discuss the related work in Section VI and conclude the paper in Section VII.

II. Making Cloud-Assisted Networks a Reality: Background and Market Analysis

A cloud-assisted network leverages the cloud VMs to host the virtual routers and the underlying cloud network infrastructure for its data transfer. In this section, we look at these counterparts from both economic and technological perspectives, and build a foundation for $NetUber$ design decisions based on our observations on cloud pricing trends.

A. Spot Markets

Large-scale dynamic overlays such as $NetUber$ require many cloud instances and a billing model that charges overlay operators for their actual usage of cloud resources. While the cloud providers offer per-second billing[1], the price of on-demand instances remains a concern for overlay operators. Despite a constant price reduction [18], building an overlay network with on-demand instances turns out to be more expensive than using traditional connectivity providers for the same Service Level Objectives (SLOs). Currently, largest cloud providers offer a cheaper and appealing alternative to on-demand instances. The so-called *spot instances* are spare computing resources, exactly identical to their on-demand counterparts, which are offered at a much lower price but can suddenly be interrupted by the cloud provider with a short notification. Amazon estimates that using spot instances can save up to 90% compared to the on-demand pricing [10]. Similarly, GCP preemptible instances [29] provide a flat rate of 80% of savings, and the recently announced Microsoft Azure low-priority instances [41] are expected to offer the same, compared to their respective on-demand offerings. Thus, we propose to use spot instances to minimize the cost as much as possible, while providing the same availability and performance guarantees.

In contrast, "reserved" instances refer to the cloud instances that are leased for an extended period such as 1 - 3 years, for a discounted price. Often reserved instances offer similar savings to spot instances. For example, 82% savings in Azure. However, they are unsuitable due to their time commitment that is unfit for the dynamic nature of the network demand.

Availability zones. Cloud data centers are present in several geographical locations that are known as the "cloud regions". Each region further consists of multiple "availability zones" that are physically separated servers in a given region. Availability zones provide resilience and fault-tolerance to the cloud regions. Each EC2 region has multiple availability zones that are isolated but connected to each other through low-latency links internal to the EC2 region network. Price fluctuations of EC2 spot market are inevitable and more vigorous for a few instance types in some availability zones at specific time frames. Even identical spot instances of different availability zones, inside the same region, often have different prices at times. In contrast, GCP preemptible instances have a fixed pricing unlike the dynamic pricing of AWS spot instances.

Price fluctuations. We monitored the availability, performance, and price fluctuations of EC2 spot instances over a time frame of three months (April - June 2017). Throughout our experiments, Linux r4.8xlarge spot instances remained the cheapest, yet memory-optimized EC2 instances with 10 GbE network interface. Each of these R4 instances consists of 32

[1]AWS and GCP moving to per-second billing from their per-hour and per-minute billings, starting from October 2017.

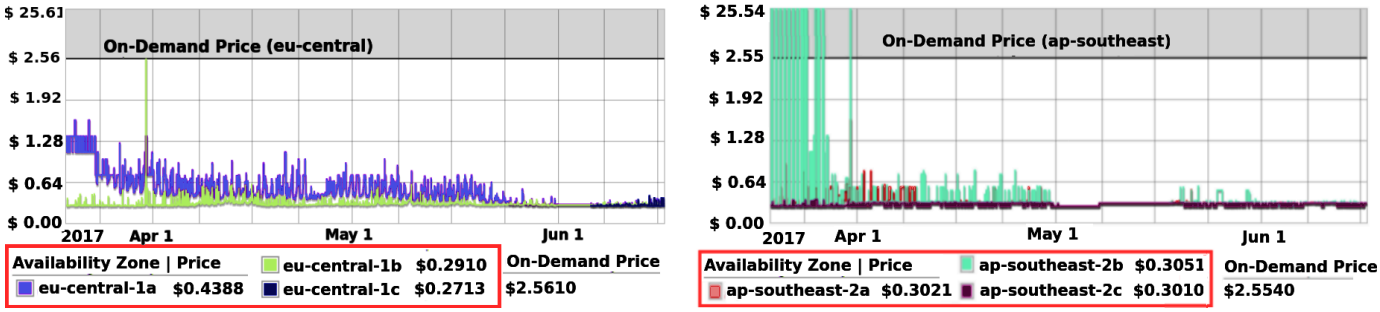


Fig. 1. AWS Linux r4.8xlarge Spot Instance Price in Frankfurt and Sydney

virtual CPUs and 244 GB of memory. They offer 10 Gbps bandwidth inside an AWS placement group (a logical grouping of EC2 instances inside an availability zone as a cluster). Network transfers outside a placement group are limited to 5 Gbps [13]. Figure 1 depicts the price fluctuations among the Linux r4.8xlarge instances of the availability zones of Frankfurt and Sydney regions during the period. We observed up to 89% of savings with spot fleets of r4.8xlarge instances in Sydney, Frankfurt, and North Virginia regions.

The spot price for Frankfurt remained relatively low and stable across all the availability zones. However, the spot price even exceeded the on-demand price in Sydney for availability zone 2b at times, while the other two availability zones remained cheaper for the spot instances. Instances of the zones eu-central-1c and ap-southeast-2c had a relatively steady and cheap price. While it is straightforward to opt for instances from the availability zones that have remained cheaper recently with a stable price, fluctuations in the future are unpredictable. Hence, while some spot instances belonging to a particular availability zone are being terminated, spot instances may be spawned in the other availability zones of the same region. This dynamic nature of the network poses the question of how the inter-region traffic should be re-adjusted to route to the current active instances.

Multiple spot instances can be spawned at once adhering to user specifications in target capacity and cost threshold, through approaches such as EC2 Spot Fleet [12] to mitigate the problem of scale and complexity in managing a large number of instances at once. Currently, EC2 spot instances are also available with predefined duration from one to six hours, 30 - 50% cheaper than the on-demand instances, making them less volatile and more reliable [10]. We observed that by leveraging multiple availability zones in each region, a stable overlay could be operated using the cheapest instances over time.

B. Cloud Data Transfer

Inter-region cloud data transfer prices remain relatively high as there is no "spot data transfer" that provides cheaper data transfer with a volatile bandwidth. The data transfer pricing exhibited a slower decline compared to that of IP transit. Around 20% and 25% of price reductions have been reported in 2010 [16] and 2014 [17] respectively, for data transfer out from the EC2 instances.

While it is typical for small enterprises and home users to connect to the cloud servers through the public Internet, for large-scale data transfers, a dedicated connection or co-locating with a cloud Point of Presence (PoP) is recommended for throughput and cost efficiency. Such a direct connection avoids depending on a third-party connectivity provider such as ISPs, which incur more costs and also limit the scale of data transfer (for example, typically ISPs offer up to 1 TB per month for home users abiding by the data rate promised in the data plan). Cloud data transfers through private direct connections thus avoid the bottleneck caused by the Internet-based connectivity between the user data centers and cloud servers. The virtual network overlay users must have an existing connection to the cloud provider or set it up directly with the cloud provider or its partners. The costs of setting up the Direct Connect is pay-per-use and not more expensive than the alternatives with the same throughput. It is configured and paid directly by each cloud user to the cloud provider.

Currently, cloud providers such as AWS and GCP do not charge for the data transfer into an instance either from the Internet or the other regions. They charge for data transfer out of a region, which differs based on the destination: the Internet, a server connected by Direct Connect, and to any other region. Currently, AWS typically charges the inter-region traffic independent of the destination region (with a few exceptions for nearest regions such as cheaper transfer between the US East regions - North Virginia and Ohio). On the other hand, GCP clusters the regions into four groups (worldwide, Asia, China, and Australia) to give differentiated rpricing based on the group of egress/destination region.

Cloud region price disparity. Figure 2 illustrates the disparity of pricing among the AWS regions to transfer a unit of data (1 TB), for transfers up to 10 TB. Larger transfers become cheaper per unit, with the price reaching almost half when the total volume reaches 500 TB. For example, it decreased from 92.16$/TB to 51.20$/TB for transfers out from regions 1 - 8 (Canada and EU-based regions, and US-based regions except for GovCloud) to the Internet. The cloud data transfer options such as Direct Connect are cheaper than sending data from the cloud to a customer server directly through the Internet. The discrepancy in pricing among the regions is visible (which is comparable to the IP transit price disparity); regions 1 - 9 (US, Canada, and EU) remain much cheaper than the others.

We observe that the current pricing of data transfer for the cloud providers is not supportive of a more comprehensive adoption of an Internet-scale cloud-assisted network, due to the lack of a differentiated pricing model based on the current lo-

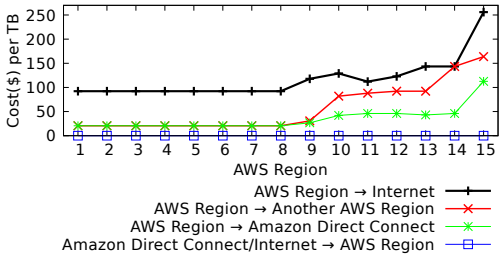


Fig. 2. Data Transfer Cost for AWS

cal time or demand for bandwidth. Moreover, cloud providers charge for the data transfers by the volume of data transferred, rather than by the data rate, unlike transit providers or ISPs. Regardless of the throughput, the cloud user pays the same amount based on the amount of data transferred. Therefore, there is no incentive for the cloud users to opt for a slower data rate even when their application is delay-tolerant. Furthermore, not considering the cloud overlay network scenario, cloud providers discourage long-distance inter-region data transfer to counter communication delays due to poor SaaS design.

Based on our observations and subsequent analysis on the cloud data transfer offerings and the availability of cheaper spot VMs, we deduce that the data transfer will contribute with the most significant share to the expenses for the Internet-scale overlays deployed over multiple regions. Thus data compression and minimizing data transfer path lengths can enable a cost-efficient execution of the cloud-assisted network.

III. Towards NetUber Deployments

In this section, we will look into the deployment architecture of three primary use cases of $NetUber$: i) a cheaper point-to-point connectivity between two regions for data transfers, ii) a premium connectivity between multi-cloud regions for end users for faster data transfer and better SaaS performance, and iii) a connectivity provider with additional network services.

A. Economical Point-to-Point Connectivity

$NetUber$ leverages spot instances to offer short-term or small-scale direct access between two geographically separated endpoints, as an economical alternative to enterprise MPLS networks. $NetUber$ has no dedicated servers. Its broker is hosted in a set of spot instances per region. Cloud monitors such as AWS alarms are leveraged to ensure that each region has at least one broker instance that is active and not scheduled for termination. For a stable overlay, we need some instances in each region, based on the bandwidth demand and the number of active instances at any given time. At any moment, the broker purchases instances from the availability zone that has the cheapest of the higher performance instances in a region. Over time, instances are spawned and maintained across multiple availability zones. Therefore, the expensive ones to maintain are terminated at the earliest.

Each $NetUber$ cloud instance hosts a virtual router. Each virtual router can dynamically connect to the virtual routers of certain spot instances of another region through the overlay, based on the users' connectivity or data transfer requests. Consider a scenario where a customer chooses to send data from her server s_o to the destination server s_d. These servers are in the cloud provider regions r_o and r_d respectively and are connected to the cloud provider through a dedicated connection such as Amazon Direct Connect. Figure 3 illustrates a representation of a sample $NetUber$ deployment that offers a direct connection between s_o and s_d. $NetUber$ is composed of many spot VMs in multiple regions, connected through the overlay network of virtual routers.

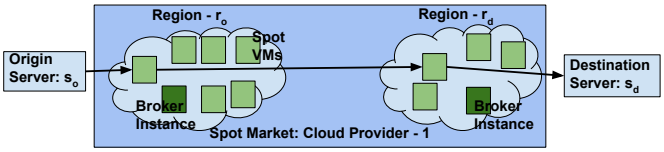


Fig. 3. $NetUber$ Deployment with a Single Cloud Provider

The broker instances monitor the resource utilization of the current VMs purchased in the spot market. They alter the spot fleet policies to bid for more instances, when the existing VMs are not sufficient (measured with a margin, to avoid performance degradation) to address the demand for connectivity and when the price for the spot instances are profitable to $NetUber$. The technical challenges include, i) initializing a newly spawned instance to operate as a virtual router in a short time, and ii) ensuring that the instances can be connected and identified through an overlay, other than their physical address, as spot instances remain volatile. The list of spot instances of the virtual routers in each region can be provided through an accessible and reliable location in the cloud provider such as an S3 bucket. The broker handles the updates to the list of instances, consisting of new and terminated spot instances.

B. Higher Performance Point-to-Point Interconnection

On the intra-domain traffic, an ISP can seek the shortest path as it controls the network. Since Border Gateway Protocol (BGP) decisions are mainly policy-oriented, it may not result in the selection of the best or the shortest path. With the cloud instances, $NetUber$ can choose to intelligently route the traffic towards the VMs in the exact regions, minimizing the number of hops and path length.

Currently, a few cities and geographical regions host the cloud regions for multiple providers. For example, North Virginia, Mumbai, London, São Paulo, Tokyo, and Singapore host both AWS [11] and GCP [28] regions. As of now, Ohio, North Carolina, Seoul, Canada central, and Ireland are AWS regions but not GCP regions; Iowa, Belgium, South Carolina, and Taiwan are GCP regions that are not AWS regions.

Figure 4 elaborates this scenario with the cloud provider 1 having presence in regions r_o and r_i, and the provider 2 with presence in regions r_i and r_d. None of the providers are present in both r_o and r_d, while both providers are present in r_i. $NetUber$ functions as a mediator between the two cloud providers to enable data transfer from $r_o \rightarrow r_d$ by interconnecting between the cloud providers in r_i, through the Internet-based connectivity or a direct/dedicated connectivity offered by the cloud providers between the clouds.

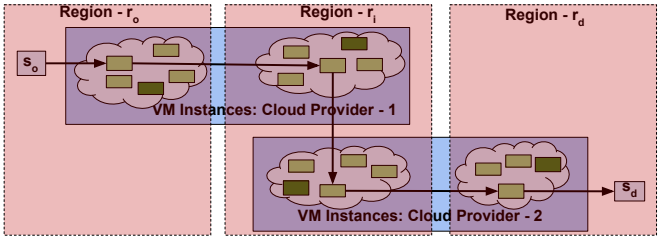


Fig. 4. Deployment Across Multiple Cloud Providers

The latency of the $NetUber$ inter-cloud interconnection remains minimal as multi-clouds of the same region are proximate to each other: they may share the same co-location facilities, or potentially be interconnected via a direct connection between the servers of the cloud vendors. For example, AWS Direct Connect can offer a direct interconnection between a pair of AWS and GCP instances in r_i. This architecture provides a higher performance point-to-point connectivity for the end users for data transfers to a geographically remote region, instead of connecting directly through an ISP.

$NetUber$ needs to consider operational differences between the cloud vendors for a stable execution. For example, currently, an AWS instance will be terminated by AWS when the current spot price exceeds the bid, with a 2-minute notice. GCP offers a 30-second notification. An AWS spot instance is terminated either by the user or by AWS when the current spot price exceeds the user bid price or when the spot resource pool in an availability zone is over-utilized. GCP terminates every spot instances 24 hours after they were started, in addition to the same conditions as AWS spot instance termination. $NetUber$ avoids shutting down instances on its own for the sake of stability, except for terminating the instances after supporting a significant spike in bandwidth demand.

A SaaS provider can use $NetUber$, instead of having geo-replicated deployments in multiple cloud regions which can be technically more challenging and more expensive. Furthermore, $NetUber$ supports more regions beyond those supported by any single cloud provider. For example, SaaS applications hosted in r_d can be accessed by an end user in r_o through $NetUber$ more reliably than through the public Internet. Thus, a SaaS provider can exploit $NetUber$ to create a point of presence in multiple regions while hosting the application in just a single region. Hence, $NetUber$ can be a potential cost-efficient alternative to geo-replicated solutions.

C. A Provider of Network Services

Virtual Network Functions (VNFs) such as packet scrubbers, transcoder, firewalls, load balancers, and proxies, can be hosted in the spot VMs of $NetUber$ as SaaS to perform middlebox actions to alter the data flow transferred atop the overlay. For example, forwarded data can be encrypted or compressed at an instance before the inter-region transfer, if prompted by the user, as additional services. Encryption enhances the security of data transferred, while compression allows an economic transfer, with minimal latency as data can be compressed in-memory in the spot instances. We can host caching services in $NetUber$ instances to optimize or limit WAN traffic. $NetUber$ can also be used for content distribution atop the overlay or mitigation of distributed denial of service (DDoS) attacks on the customer networks.

Hosting VNFs and SaaS on top of an overlay such as $NetUber$ is straightforward as these applications directly consume the cloud resources. As elaborated in Section II, $NetUber$ relies upon highly-optimized 10 GbE spot instances (i.e., R4), which are ideal to support all the above computation-intensive network functions, as opposed to smaller unstable spot instances. These optimized instances have abundant memory (244 GB memory each) and CPU resources. Thus, with a relatively stable memory, computing, and networking resources across the regions, $NetUber$ can be used as a framework for third-party network services on a cloud platform. We omit elaborations on such services for the sake of brevity.

IV. Economic Feasibility of a Cloud-Assisted Virtual Connectivity Provider

Various pricing models have been proposed for connectivity providers [33], content providers [30], and clouds [38]. Cloud vendors list their VM prices at an hourly rate though they charge per second. $NetUber$ follows the same pricing scheme since it acquires its instances on a per-second basis. Since the connectivity providers list their charges per-month, we assume one month as the total time in our models, for a fair comparison. While a cloud-assisted network provider may be able to negotiate a discounted price with the cloud providers for a large-scale spot resource acquisition, we limit ourselves as regular users for the sake of a realistic evaluation. A cloud vendor itself may choose to operate a $NetUber$-like overlay. However, our interest is limited to the decoupling of the overlay provider from the cloud provider.

Equation 1 defines $\lambda_{o,d}$ as an end-to-end unit data transfer cost from s_o to s_d. Currently, cloud providers do not charge for incoming data from the Internet or another region. The $NetUber$ customers incur a cost (charged per used port-hours, at an hourly rate, by AWS Direct Connect), D_o and D_d, to connect the servers s_o and s_d to the cloud provider. Thus,

$$\lambda_{o,d} = \lambda_{s_o,r_o} + \lambda_{r_o,r_d} + \lambda_{r_d,s_d} = D_o + \lambda_{r_o,r_d} + \lambda_{r_d,s_d} + D_d \quad (1)$$

By substituting the values for the two (also can be generalized for more than two) cloud providers, Equation 1 can be extended to include the multi-cloud scenario depicted in Figure 4. $\lambda^{(1)}$ and $\lambda^{(2)}$ denote the unit data transfer costs by the cloud providers 1 and 2 respectively. The instance of the *cloud provider* 2 in r_i is just an external server connected through the public (Internet-based) or a dedicated direct connectivity for the *cloud provider* 1, whereas it functions as the origin cloud server from the perspective of the *cloud provider* 2. Thus, the cost associated with the *cloud provider* 1 is denoted by $\lambda^{(1)}_{r_o,r_i} + \lambda^{(1)}_{r_i,s_i}$, whereas the cost associated with the *cloud provider* 2 is denoted by $\lambda^{(2)}_{r_i,r_d} + \lambda^{(2)}_{r_d,s_d}$. D_i denotes the cost of connecting the instances of both cloud providers at the region i. Hence, Equation 2 denotes the total cost.

$$\lambda^{(1,2)}_{o,d} = D_o + \lambda^{(1)}_{r_o,r_i} + \lambda^{(1)}_{r_i,s_i} + D_i + \lambda^{(2)}_{r_i,r_d} + \lambda^{(2)}_{r_d,s_d} + D_d \quad (2)$$

We formulate the total cost from all the vendors for $NetUber$ C, in Equation 3. C consists of the cost associated with acquiring the spot VMs and the cost of data transfer. $c_{v,r,i,t}$ defines the cost for an instance from the cloud vendor ($v \in V$) from a region at a given time step between t_0 and t_f. Since the spot price continues to fluctuate, the cost to acquire the required number of spot instances ($i \in I$) in each of the regions ($r \in R$) is calculated as a time integral over its execution time, and summed for all the instances from each region of all the cloud vendors.

The data transfer cost is billed by the cloud provider per the volume of data transferred. Therefore, it is calculated by a time integral of data rate b_t through a cloud path to its completion. Since $\lambda_{o,d}$ denotes the unit data transfer cost involving all the cloud paths, we calculate the total data transfer cost from the first $NetUber$ instance that receives the user traffic, $\forall i \in I_{r_o}$, for each region of all the cloud vendors.

$$C = \sum_{v \in V} \sum_{r \in R} \left[\sum_{i \in I} \int_{t_0}^{t_f} c_{v,r,i,t} dt + \sum_{i \in I_{r_o}} \int_{t_0}^{t_f} (\lambda_{o,d} b_t) dt \right] \quad (3)$$

We observed that the data rate of the inter-region data transfers b_t is proportional to the network interface (β) of the instance. β is 10 Gbps in the r4.8xlarge instances used by $NetUber$. However, it is impossible to reach the full network interface capacity in the inter-region data transfer. Cloud data transfers between regions have a degradation from the promised network interface. We define the degradation in the data rate of inter-region data transfer as a ratio of the network interface of the pair of VM instances, $\chi_t \in (0, 1)$. The actual data rate $b_t = \beta \times (1 - \chi_t)$.

Data compression at the source can significantly reduce the costs, given that cloud providers do not charge for the incoming data. Many cloud compression tools, general purpose or optimized for specific file formats, make lossless compressions feasible at the time of the cloud transmission [46]. By compressing the data before the inter-region transfer, we can significantly increase throughput or the actual data transferred per unit time. We define a compression ratio γ_t as the percentage of size reduction from compression without incurring data loss. γ_t and χ_t vary with time, unpredictable to $NetUber$. Equation 4 illustrates the effective data rate b.

$$b = \frac{b_t}{1 - \gamma_t} = \frac{\beta \times (1 - \chi_t)}{(1 - \gamma_t)}; \chi_t, \gamma_t \in (0, 1) \quad (4)$$

$NetUber$ proposes to charge its customers based on their requested bandwidth (b), the length of the bandwidth usage (τ), and a direct unit (per time unit, per unit data rate) cost ($\Lambda_{o,d}$) to acquire the instances and data transfer from the cloud provider. c_{v_*,r_i} defines the cost of acquiring intermediate instances from any vendor v_*. c_{v_*,r_i} is 0 if v_o = v_d, as this makes $NetUber$ overlay with just one cloud vendor. β defines the network interface of the instance. To find the instance cost per unit data rate, we divide the cost of instances by the capacity of their network interface. Thus, $NetUber$ defines the charge Γ for its customer as a cost function (Equation 5) such that it remains profitable, with the total income of $NetUber$ from all its users for their connectivity demands remain higher than its cost of acquiring spot instances and data transfer costs.

$$\Gamma = f(\tau, b, \Lambda_{o,d}), \textbf{where} \quad (5)$$

$$\Lambda_{o,d} = \beta^{-1} \times (c_{v_o,r_o} + c_{v_d,r_d} + \sum_{if(v_o <> v_d)} c_{v_*,r_i}) + \lambda_{o,d}.$$

V. Evaluation

In this section, we aim to answer two questions: i) when is $NetUber$ more cost-efficient than connectivity providers?, and ii) how does the performance of $NetUber$ compare to using direct Internet paths?

Prototype deployment. We deployed a prototype of $NetUber$ on multiple r4.8xlarge optimized spot instances (each with 10 GbE network interface) in each AWS region, as virtual routers. We performed an initial evaluation of various origin and destination regions. We reached around 50 Mbps between two instances in any regions with a single TCP connection. We achieved 1.2 Gbps of maximum stable inter-region bandwidth between the pair of 10 GbE instances with parallel connections. Thus, we deployed our prototype on at least 9 pairs of r4.8xlarge spot instances to achieve 10 Gbps bandwidth between two regions. We confirmed that the maximum bandwidth (1.2 Gbps) obtained was independent of the origin and the destination regions. By choosing a spot instance r_o in the overlay and placing it along with the origin server s_o in the same placement group, $NetUber$ ensures that s_o utilizes the complete 10 GbE in sending data to the overlay. Since spot instances are billed per second, we spawned the instances only when necessary. Instances are shared across users for multiple data transfers (at the same time, or at different time intervals, based on the data rate required for the transfer). Thus, the entire data rates of the instances are exploited, with minimal underutilization.

Infeasibility of using smaller instances. There are alternatives to the R4 instances that we used in $NetUber$: smaller spot instances and on-demand instances. The smaller instances such as C3 instances have a moderate network. We found two issues with these moderate network instances: i) we need to acquire a lot of them, which is more complicated to maintain due to the need for a substantial number of parallel connections, and ii) they are very unstable. We noted that with the cost of acquiring the number of 10 GbE instances, we could have around 2 - 4 Gbps, yet unpredictable, inter-region bandwidth with numerous moderate-network spot instances. However, the moderate network instances offered no promised guarantees for the bandwidth, unlike the R4 instances (r4.8xlarge and r4.16xlarge have 10 GbE and 25 GbE interfaces, respectively).

We were able to obtain the R4 spot instances promptly while having to wait for up to one hour for the moderate network instances. The r4.8xlarge of $NetUber$ instances stayed alive throughout the experiments which lasted up to 3 months, while smaller instances shut down frequently. Thus, we observe that it is possible to have a stable overlay over the 10 GbE spot instances, whereas currently, it is not feasible over the smaller ones. There was no difference in the quality of paths between

the on-demand and spot instances. Thus, we observed that using smaller on-demand instances provide worse data rate or a much higher cost than using the 10 GbE spot instances for a functional prototype.

A. Economical Alternative to Connectivity Providers

To evaluate the cost efficiency of $NetUber$, we compare the data transfer cost of $NetUber$ from Frankfurt to Sydney, against the offerings of 2 connectivity providers in the EU/US regions, marked as CP-1 and CP-2 in Figure 5. Due to cost restrictions, our extensive study only covers this pair of EC2 regions. However, based on the past approximate spot instance pricing details we gathered, we believe that our approach can be generalized to other pairs of regions. We also consider a compressed data transfer with $NetUber$ in the evaluations, accounting for the potential compression-as-a-service deployment in the $NetUber$ instances. We report the cost of the instances as the average price during the period.

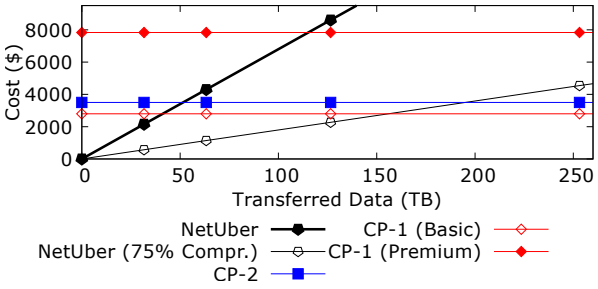


Fig. 5. Monthly Fee for 10 GbE Flat Connectivity

CP-1 is an infrastructure provider with an extensive, geographically-distributed infrastructure that offers connectivity as an alternative to transit providers. It provides two options: a basic scheme to connect to regular networks choosing the cheapest paths, and a more expensive premium scheme to connect to premium networks (connecting with large IXPs and premium networks) for better throughput and shortest paths. CP-2 is a transit provider. We obtained these price quotes via private email queries. As these quotations are not public (as typically transit providers do not list the prices in public), we must refrain from disclosing the providers. We include the costs of acquiring instances, data transfer costs, and the AWS Direct Connect cost for a continuous data transfer of the given volume for $NetUber$.

Up to 75% of lossless compression has been reported in compressing streaming data in real time [1]. We demonstrate a similar (0.75) or higher γ_t with in-memory compression at r_o. With these values, b = β · (0.12) / (1 - 0.75) = 0.48 · β = 4.8 Gbps, from Equation 4. While the inter-region data transfer achieves only 0.12 · β (or 0.48 · β with compression), AWS Direct Connect reaches the complete data rate of β. In Figure 5, we plot the minimum price of $NetUber$ as the cost charged by EC2 for the spot instances and the data transfer. Here we consider both regular (γ_t = 0; χ_t = 0.88) and compressed (γ_t = 0.75; χ_t = 0.88) $NetUber$ transfers.

3164.0625 TB per month (10 Gbps = 10/8 · 3600 · 24 · 30 GB/month) of data can be transferred between a pair of instances with 10 GbE interface. For volumes of data transfers up to at least 50 TB, $NetUber$ always offered a competitive price (compared to the benchmarked connectivity providers) and remained globally profitable. When 75% compression is assumed, $NetUber$ was still cheaper up to 200 TB. Thus, by leveraging the availability of abundant memory in the r4.8xlarge instances, we can employ enhancements profitable to $NetUber$ or additional VNFs that can be executed as value-added services on the data traffic.

TABLE I
PING TIMES (MS): REGULAR INTERNET VS. $NetUber$

Origin → Destination	**Direct**	***NetUber (via)*** Improvement
Vladivostok → São Paulo	362.72	307.08 *(Tokyo)* 15.34%
Hobart → Mumbai	347.22	248.41 *(Sydney)* 28.46%
Seoul → São Paulo	321.72	299.31 *(Seoul)* 6.97%
Tashkent → Singapore	351.61	258.57 *(Mumbai)* 26.46%
Nairobi → Tokyo	403.87	386.37 *(Mumbai)* 4.33%
Frankfurt → Tokyo	296.87	237.34 *(Frankfurt)* 20.05%
Thuwal → Tokyo	346.01	324.30 *(Frankfurt)* 6.27%
Prague → São Paulo	224.90	221.40 *(Frankfurt)* 1.56%
Nuuk → Sydney	415.02	352.46 *(Canada)* 15.07%
Fairbanks → Mumbai	441.57	435.64 *(Canada)* 1.34%
São Paulo → Paris	239.45	210.72 *(São Paulo)* 12.00%
Tacuarembó → Montreal	203.42	186.01 *(São Paulo)* 8.56%

B. Higher Performance Point-to-Point Interconnection

$NetUber$ is not always cheaper. But can it perform better when it is equally or more expensive than using the standard Internet-based connectivity? To assess the performance, we compare the round-trip time latency (ping time) between two endpoints that connect through $NetUber$ against the latency using Internet-based connectivity of ISPs.We sent pings between the endpoints, first through the standard connection, and then via $NetUber$ by entering the overlay through the nearest AWS region. For the geographically distributed servers, we used RIPE ATLAS Probes [15] and our physical servers.

We benchmark $NetUber$ along with ISPs for faster Internet routes, against using just an ISP, how the chosen ATLAS Probes are connected by default to the Internet. We repeated the evaluations ten times and listed the average ping times (in milliseconds) in Table I, along with the AWS region that the ping is routed through for $NetUber$, as well as the improvement when using $NetUber$. We measured the performance improvement by the drop in latency, leaving other properties such as jitter and loss rate for future work. In all the cases, we observe that going through $NetUber$ overlay offered better latency (up to 30% improvement) than directly connecting through the ISP, as long as a cloud region exists relatively near to the origin server, en route to the destination. The ISP-based connectivity remained the bottleneck in throughput as it reached up to only 50 - 75 Mbps.

The results indicate that even without dedicated connections to the cloud provider, an ISP user can resort to $NetUber$ for better latency and throughput for data transfer and accessing SaaS hosted in a far cloud region, rather than directly connecting through the user's ISP. However, we predict a better latency and throughput when a dedicated connection such as Amazon Direct Connect connects the servers to the overlay. Similarly, $NetUber$ can also be used in conjunction with

FTTH and community-based initiatives [14] for faster Internet routes, as they are not widespread. One can even use $NetUber$ adaptively, such that only the transfers with a cloud region en route to offer better latency go through the $NetUber$ overlay. We leave further discussion on these for future work.

VI. Related Work

A. Cost Efficiency with Cloud Infrastructure

Cloud-based infrastructures can increase the cost-efficiency of interconnections while minimizing the deployment time. Voxility [44] leverages its vast distributed infrastructure to provide network services and end-to-end interconnection, cheaper than the transit providers with more flexible agreement options for short-term and small-scale interconnections. CloudDirect [22] offers auxiliary services such as backup and disaster recovery atop cloud offerings. Cloud resources of $NetUber$ can be leveraged for more than just connectivity, including network services such as caching, content distribution to multiple local subscribers, and data analytics over wide-area networks [43].

Various approaches have been proposed, to reap the economic benefits, while addressing the technical challenges inherent to the volatile nature of spot instances. A third party or a broker leveraging resources from multiple cloud providers, and reselling them in a vertical market, has been found to be beneficial for both the broker as well as the cloud providers [36]. This multi-cloud infrastructure is in line with that of $NetUber$. Dynamic bidding policies are developed to support deadline-constrained jobs in spot instances [49]. Temporal multiplexing of burstable instances (to have a constant higher availability of CPU cycles) and spatial multiplexing of spot instances (to have reliable connectivity with redundancy in the path) can be performed inside a single AWS region with minimal overhead [26]. A trusted third party such as Google Fi [9] can function as a virtual ISP by exploiting the resources of multiple ISPs [50]. However, no comprehensive study has been conducted to realistically determine the feasibility of a virtual ISP that leverages the resources of spot instances, as a regular cloud user. $NetUber$ exploits the differentiated pricing of various availability zones for a relatively stable overlay.

Bidding in the spot markets of multiple regions can minimize the costs of CPU-intensive workloads, increasing the availability of the Internet services [34]. While $NetUber$ bids in multiple regions to acquire VMs in geographically distributed locations to host the virtual routers, it cannot use migrations between VMs in different regions for cost efficiency, as it will, in turn, increase the bandwidth consumptions and the number of hops. Cloud brokerage services have been built on spot instances with scheduling and reservation mechanisms, to minimize computing costs for jobs with a strict deadline, up to 57% [47]. Cost efficiency and performance of in-memory caches have been improved in the cloud, by deploying in spot instances while exploiting burstable instances for a backup [45]. But the scope of those research work is limited to computing, while $NetUber$ focuses on network connectivity, data transfer, and additional VNFs on the path.

B. Decoupling the Internet

Software-Defined Internet Architecture (SDIA) [39] decouples the architecture of the Internet from the infrastructure, by modifying the way interdomain tasks operate, through SDN and MPLS. SDIA and $NetUber$ share the goal of connecting endpoints on the Internet regardless of the underlying infrastructure. However, $NetUber$ focuses on network virtualization and does not alter how the underlying physical network works. Consequently, $NetUber$ can be deployed on existing cloud providers without any modifications to the cloud networks. Jingling [27] separates the network functions outside the network towards external "Feature Providers". While Jingling delegates network functions to third parties, $NetUber$ virtualizes the entire network with virtual routers running atop spot VMs, by a third party broker. Both $NetUber$ and Jingling do not have control over the exact physical location of the system. Thus, specifying policies of the end users and identification of the cloud instances should be done through service layer instead of the physical address.

Virtual connectivity providers that do not control the infrastructure have been proposed, following an approach similar to that of $NetUber$ [25], [48], [21]. Cabo decouples ISPs into infrastructure providers and service providers, with concurrent networks that run multiple virtual routers atop each physical router, thus virtualizing links between any two virtual nodes [25]. Slicing the home networks can enable various service providers to reduce the costs and overhead associated with deployment and management, by sharing a common infrastructure [48]. However, $NetUber$ focuses on leveraging the cheap spot instances, and thus offers an economical approach to deploy on a large scale.

There have been industrial efforts following the same goal with $NetUber$, aiming at a fast direct interconnection between two endpoints, without relying on traditional connectivity providers in the region. PacketDirect [7] is an SDN-based platform that reduces the time to set up interconnections through its SDN-based framework. MPLS providers such as iTel [5] connect multi-location decentralized offices with a private layer-2 network, a unified connection to the whole organization. These providers differ from the traditional MPLS networks that merely provide connectivity between two endpoints, thus still requiring Ethernet connections for each office.

VII. Conclusion

Connectivity providers limit their agreements regarding minimum length and scale, preventing customers with short-term (in the scales of minutes, opposed to months), or minimal bandwidth requirements. $NetUber$ aims to address this as a virtual connectivity provider, built as a cloud-assisted overlay, running atop spot instances purchased from cloud providers for a low price. $NetUber$ aims for affordable end-to-end network connectivity for anyone, while not owning the infrastructure. In this paper, we built a case on why a virtual connectivity provider without any fixed resources may not just be technologically feasible, but also be economically sustainable.

We presented case studies with $NetUber$ as i) an economical alternative to connectivity providers for data transfers up to 50 TB, ii) a higher performance connectivity as an alternative to ISPs for end-to-end inter-region data transfer and accessing SaaS hosted in far regions without geo-replication, and iii) a provider for network services on top of a cloud-assisted overlay. Our analysis of the spot instance prices shows that cloud-assisted overlay network costs depend on a variety of factors, including geographical locations and current demand. We observe the enhancements in performance in comparison to ISPs and cheaper data transfer for small decentralized enterprises. As a future work, we envision an Internet-scale economic analysis and deployment of $NetUber$ on top of multiple cloud infrastructures.

Acknowledgements: We are thankful to Márcio Santos and Virgil Truica for their insights. This research was supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960), and national funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013.

REFERENCES

[1] Compression as a service is now available in the cloud! boostedge.net for isps, telcos and enterprises ubfast.com for end-users!, 2013. Available at http://www.businesswire.com/news/home/20130207006266/en/Compression-Service-Cloud!-Boostedge.Net-ISPs-Telcos-Enterprises.

[2] Encryption as a service, 2014. Available at http://www.cloudlinktech.com/wp-content/uploads/downloads/2014/07/Data-Encryption-as-a-Service_R1.pdf.

[3] Console - the cloud connection company, 2017. Available at https://www.consoleconnect.com/.

[4] Epsilon Telecommunications Limited – Connectivity Made Simple, 2017. Available at http://www.epsilontel.com.

[5] iTel MPLS (IP VPN) High Performance Connectivity, 2017. Available at http://signup.itel.com/hubfs/Sales_Resources_InfoSheets/iTel_MPLS_Info_Sheet.pdf.

[6] Megaport, 2017. Available at http://megaport.com/.

[7] PacketDirect, 2017. Available at https://www.packetfabric.com/packetdirect/.

[8] Packetfabric, 2017. Available at https://www.packetfabric.com/.

[9] Project fi, 2017. Available at https://fi.google.com/about/.

[10] Amazon. Amazon ec2 spot instances, 2017. Available at https://aws.amazon.com/ec2/spot/pricing/.

[11] Amazon. Aws regions and endpoints, 2017. Available at http://docs.aws.amazon.com/general/latest/gr/rande.html.

[12] Amazon. How spot fleet works, 2017. Available at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html.

[13] Amazon. Placement groups, 2017. docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html.

[14] B4RN. Broadband for the rural north, 2018. https://b4rn.org.uk/about-us/our-network/.

[15] V. Bajpai, S. J. Eravuchira, and J. Schönwälder. Lessons learned from using the ripe atlas platform for measurement research. *ACM SIGCOMM Computer Communication Review*, 45(3):35–42, 2015.

[16] J. Barr. Aws outbound data transfer prices reduced by $0.02/gb, 2010. Available at https://aws.amazon.com/blogs/aws/aws-data-transfer-prices-reduced/.

[17] J. Barr. Aws data transfer price reduction, 2014. Available at https://aws.amazon.com/blogs/aws/aws-data-transfer-price-reduction/.

[18] J. Barr. Aws blog. category: Price reduction, 2017. Available at https://aws.amazon.com/blogs/aws/category/price-reduction/.

[19] S. A. Baset. Cloud slas: present and future. *ACM SIGOPS Operating Systems Review*, 46(2):57–66, 2012.

[20] B. Boudreau. Global bandwidth & ip pricing trends, 2017. Available at http://www2.telegeography.com/hubfs/2017/presentations/telegeography-ptc17-pricing.pdf.

[21] C. X. Cai, F. Le, X. Sun, G. G. Xie, H. Jamjoom, and R. H. Campbell. Cronets: Cloud-routed overlay networks. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 67–77. IEEE, 2016.

[22] CloudDirect. Move to Cloud ID - quickly, easily and securely, 2017. Available at https://www.clouddirect.net/.

[23] Cloudflare. Cloudflare argo, 2017. Available at https://www.cloudflare.com/argo/.

[24] Cogent. Cogent ip transit, 2017. Available at http://www.cogentco.com/en/products-and-services/ip-transit.

[25] N. Feamster, L. Gao, and J. Rexford. How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64, 2007.

[26] A. Gandhi and J. Chan. Analyzing the network for aws distributed cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3):12–15, 2015.

[27] G. Gibb, H. Zeng, and N. McKeown. Outsourcing network functionality. In *HotSDN'12*, pages 73–78. ACM.

[28] Google. Google cloud platform - cloud locations, 2017. Available at https://cloud.google.com/about/locations/.

[29] Google. Preemptible vm instances, 2017. Available at https://cloud.google.com/compute/docs/instances/preemptible.

[30] P. Hande, M. Chiang, R. Calderbank, and S. Rangan. Network pricing and rate allocation with content provider participation. In *INFOCOM'09*, pages 990–998. IEEE.

[31] O. Haq and F. R. Dogar. Leveraging the power of cloud for reliable wide area communication. In *HotNets'15*, page 19. ACM.

[32] O. Haq, M. Raja, and F. R. Dogar. Measuring and improving the reliability of wide-area cloud paths. In *WWW'17*, pages 253–262.

[33] H. He, K. Xu, and Y. Liu. Internet resource pricing models, mechanisms, and methods. *Networking Science*, 1(1):48–66, 2012.

[34] X. He, P. Shenoy, R. Sitaraman, and D. Irwin. Cutting the cost of hosting online services using cloud spot markets. In *HPDC'15*, pages 207–218. ACM.

[35] P. Lovelock. Unleashing the Potential of the Internet in Central Asia, South Asia, the Caucasus and Beyond. *ADB Consultant's Report*, pages 27–28, 2015.

[36] A. Ludwig and S. Schmid. Distributed cloud market: Who benefits from specification flexibilities? *ACM SIGMETRICS Performance Evaluation Review*, 43(3):38–41, 2015.

[37] B. W. Norton. Internet transit prices - historical and projected, 2014. Available at http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php.

[38] R. Pal and P. Hui. Economic models for cloud service markets: Pricing and capacity planning. *Theoretical Computer Science*, 496:113–124, 2013.

[39] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *HotNets'12*, pages 43–48. ACM.

[40] D. Richman. Amazon web services' secret weapon: Its custom-made hardware and network, 2017. https://www.geekwire.com/2017/amazon-web-services-secret-weapon-custom-made-hardware-network/.

[41] M. Scurrell. Batch computing at a fraction of the price, 2017. Available at https://azure.microsoft.com/en-us/blog/announcing-public-preview-of-azure-batch-low-priority-vms/.

[42] Teridion. Teridion, 2017. Available at https://www.teridion.com/.

[43] R. Viswanathan, G. Ananthanarayanan, and A. Akella. Clarinet: Wan-aware optimization for analytics queries. In *OSDI'16*, pages 435–450. USENIX Association.

[44] Voxility. Voxility - The secure infrastructure for your amazing Cloud Service, 2017. Available at https://www.voxility.com/.

[45] C. Wang, B. Urgaonkar, A. Gupta, G. Kesidis, and Q. Liang. Exploiting spot and burstable instances for improving the cost-efficacy of in-memory caches on the public cloud. In *EuroSys'17*, pages 620–634. ACM.

[46] Y. Xing, G. Li, Z. Wang, B. Feng, Z. Song, and C. Wu. Gtz: a fast compression and cloud transmission tool optimized for fastq files. *BMC bioinformatics*, 18(16):549, 2017.

[47] M. Yao, P. Zhang, Y. Li, J. Hu, C. Lin, and X. Y. Li. Cutting your cloud computing cost for deadline-constrained batch jobs. In *ICWS'14*, pages 337–344. IEEE.

[48] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing home networks. In *HomeNets'11*, pages 1–6. ACM.

[49] M. Zafer, Y. Song, and K.-W. Lee. Optimal bids for spot vms in a cloud for deadline constrained jobs. In *CLOUD'12*, pages 75–82. IEEE.

[50] L. Zheng, C. Joe-Wong, J. Chen, C. G. Brinton, C. W. Tan, and M. Chiang. Economic viability of a virtual isp. In *INFOCOM'17*. IEEE.

Real-time Money Routing by Trusting Strangers with your Funds

Martijn de Vos and Johan Pouwelse
Distributed Systems group, Delft University of Technology, The Netherlands
Email: m.a.devos-1@tudelft.nl

***Abstract*—We explore a new stage in the evolution of digital trust, trusting strangers with your funds. We address the trust issues when giving money to others and relying on them to forward it. For fraud identification, we leverage our deployed blockchain which gradually builds trust between interacting strangers. Our blockchain fabric, called *TrustChain*, records interactions between entities in a scalable manner. This work represents a small step towards a generic infrastructure for trust, moving beyond proven single vendor platforms like eBay, Uber and Airbnb.**

Expanding upon established trust relations, we designed, implemented and evaluated an overlay network: *Internet-of-Money*. Internet-of-Money routes money to different banks through individuals, so-called *money routers*. This removes the need for central banks, to handle a payment. Our network reduces the duration of traditional inter-bank payments from up to a day and even a few days during weekends, to mere seconds. Internet-of-Money is fully decentralized, scalable and privacy-preserving.

With real-world experimentations, we prove that Internet-of-Money enables fast money forwarding. We show that the overlay network is capable of discovering a majority of available money routers within a minute. Finally, we demonstrate how profit of cheating routers is limited and that misbehaviour is punished.

I. Introduction

Creating trust between strangers is at the core of numerous successful Internet companies. Starting 22 years ago, Craigslist offered an unmoderated mailing list of advertisements and gossip on which buyer and seller could be trusted. eBay formalised this in 1997 and introduced a star-based rating system that enables traders to build a trustworthy profile [1]. The e-commerce platform was launched at a time when people were still hesitant to use their credit card on a technology called The Internet. Nowadays, people let strangers sleep in their houses using Airbnb (since 2008). We trust Uber (since 2009) with our physical security and get into cars late at night with a driver that has never undergone a criminal background check or given a government license. These influential milestones in the evolution of digital trust are shown in Figure 1.

We continue this evolution of building trust. We created an operational platform for one of the most challenging and sensitive applications, having others handle your money.

Bitcoin created money without the need for banks [2]. In the past, people were required to trust a central bank and a host of other intermediaries when making payments [3]. The fundamental technology of Bitcoin, blockchain, radically reduced the need to trust financial middlemen. It bootstrapped an economy where no one can be stopped from spending their money. Despite widespread speculation and ecosystems being worth billions, blockchain in general suffers from scalability issues due to inefficient mechanisms for fraud prevention. Bitcoin is theoretically limited to seven transactions per second and Ethereum has a throughput of around 20 transactions per second [4]. Despite various scalability efforts like proof-of-stake and sharding, broader adoption of blockchain stays out.

ISBN 978-3-903176-08-9

While a majority of Internet users trust the company behind popular platforms, the events involving Mt. Gox highlighted how digital trust can be established and compromised [5]. Mt. Gox was at one point the largest Bitcoin exchange worldwide. In 2014, hackers stole Bitcoin, worth around $460 million at that time. This event, together with major data breaches in 2017 at high-profile companies like Uber and Equifax, exposed the weakness of centralized architectures [6]. They motivate research around decentralized technologies, like blockchain.

The generic problem of building trust between strangers resides on the edge of technology, sociology and behavioural science [7]. The question whether someone can be trusted, depends on properties like personality, level of authority, culture and past behaviour. In this research, we address the trust problem from a technological perspective, using tamper-proof interactions on a scalable blockchain. This structure is built to detect fraudulent behaviour and misrepresentation. We explore whether a trust model based merely on historical encounters is sufficient to trust strangers with your money.

With established trust relations, we demonstrate how one can transfer money within seconds between different banks by relying on others to act as financial intermediaries. In comparison to most proven platforms, our solution is designed to be fully decentralized and autonomous. Our work is motivated by slow money transfers to other banks using existing systems. Inter-banking payments often take up to a day or even a few days during weekends to arrive in the account of a beneficiary.

The main contributions of this work are as follows:

1) A trust model, based on repeated interactions and stored on a tamper-proof, scalable blockchain.
2) *Internet-of-Money*, a novel overlay network that allows real-time money routing to other banks.
3) Experimental quantification of the performance of our trust model, the speed of money transfers and the efficiency of our overlay network.
4) A framework to interface with multiple banks and to initiate payments to others using Internet-of-Money.

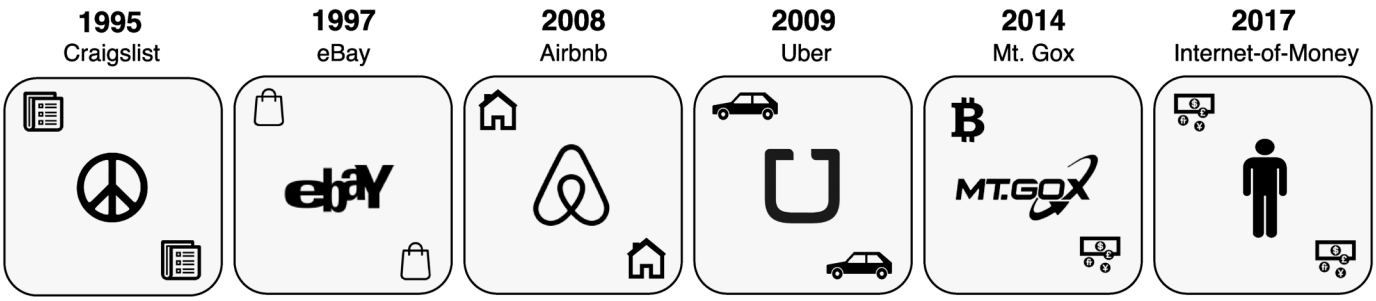


Fig. 1: Influential milestones in the evolution of digital trust.

II. Problem Description

Trust and fraud are essential problems to address when trusting others with your money. While most cryptocurrencies use a lottery system to stumble upon trustful executors, we rely on game theory to ensure honest behaviour has the largest rewards. We focus on the effective detection and punishment of fraudulent behaviour. While it is a common belief that money transfer systems should be safe against all kinds of fraud, we argue that it is sufficient for fraud to be detectable and punishable. This is comparable to the operation of credit card companies, which have to deal with a considerable amount of fraud on a daily basis. Detection of such fraud is non-trivial.

The trust problem in this work can be modelled by the prisoner's dilemma, where two entities can either cooperate or betray each other [8]. Betrayal is also called *defection*. In the iterative prisoner's dilemma, players cooperate or deflect iteratively and are able to punish opponents for their past decisions. We assume a *send and forward* model where a user first sends money to another user, who in turn forwards the money to someone else. Forwarding funds is considered cooperation whereas keeping the money is seen as defection. Detecting whether an entity has defected is a key requirement. Not cooperating should be punished by digital ostracism.

Many companies rely on centralized reputation mechanisms to manage trustworthiness of platform participants. In general, this leads to two problems. First, a solid track record built in one platform is often not reusable on other platforms. Second, building and maintaining an interaction history on multiple platforms simultaneously leads to fragmentation of one's trustworthiness scores. Users are increasingly being protected from such data silos by regulation [9]. Our aim is to devise a decentralized and generic reputation mechanism.

A mature research community exists around the design of decentralized reputation systems [10][11][12]. A notorious attack in decentralized systems occurs when a user first builds a high reputation by acting honest for some time and then abuses this accumulated trust for personal enrichment. This is also called the "pump and dump" method. Another challenging attack in decentralized networks is the Sybil Attack, where an individual creates multiple fake identities and initiate transactions with them to increase his or her standing in the community [13]. The Sybil Attack is hard to solve without trusted third parties, particularly in decentralized networks.

III. Settlement of Traditional Payments

Prior to elaborating how we can use trust and individuals to realise real-time money transfers, we briefly explore the process of performing a payment with existing infrastructure. International payment systems are often proprietary and lack transparency. The largest inter-bank communication network is SWIFT, the Society for Worldwide Interbank Financial Telecommunication [14]. In April 2017, SWIFT recorded an average of 28.38 million payments per day or around 328 per second. This legacy network was founded in the 1970s and programmed in a language from the 1950s (COBOL). While a majority of financial institutions worldwide rely on SWIFT, joining the network is an expensive and involved process. Due to the high costs when initiating cross-border payments, many users and companies are shut out of the system. It is estimated that back-office costs for international payments need to drop by 90% to 95% for banks to remain competitive [15].

The SWIFT network exposes high processing or *settlement* times for inter-bank payments, in particular for international payments. While many companies and banks are working on new platforms to enable instant (international) payments, there are various issues that should be addressed. These include real-time fraud detection and robust messaging standards [15].

A payment to another bank usually involves an intermediate settlement institution that is responsible for settling a payment between two parties [3]. This is often a central bank. The settlement institution acts as an intermediary in the payment chain and reduces settlement risks. Instead of handling a large number of payment instructions and settling them individually, settlement institutions usually aggregate outstanding payments and settle them all at once on predetermined times. This is called *net settlement* or *netting*.

While inter-bank payments take a considerable amount of time to settle, moving funds within the books of the same bank is significantly faster. This is called an *in-house payment*. In-house payments have a relatively low settlement duration, usually a few seconds, since no inter-bank communication is required.

IV. Our Money Routing Mechanism

Our mechanism to perform real-time payments is based on the observation that in-house payments are settled fast. For the banks that we tested, intra-bank money transfers are settled

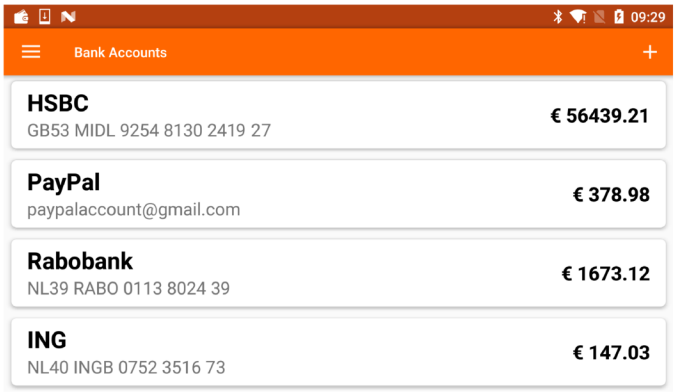


Fig. 2: Our Android application to interface with different banks and to route money in real-time.

within mere seconds (see Section VII-A). Instead of using a central bank as settlement institution, we build a network of individuals that have bank accounts with multiple banks to perform settlement on a gross basis. This works as follows: assume a Dutch buyer holding a Rabobank account, intends to pay a British merchant that holds a bank account with HSBC. When this buyer initiate a payment with existing software to the merchant, then the funds can take several days to arrive in the bank account of the merchant. However, when using an intermediary holding accounts both at Rabobank and HSBC, the buyer first sends the funds to the Rabobank account of this intermediary after which the buyer instructs the intermediary to forward the same amount of money from his HSBC account to the HSBC account of the merchant.

Since this way of sending money only involves two in-house payments, the merchant receives the funds within a few seconds. We call this process a *fast payment* We call the intermediary settling the transaction a *money router*. We use the terms *initiator* and *beneficiary* to indicate the initial sender and final receiver of a fast payment, respectively. A fast payment can be facilitated by multiple routers to increase efficiency and availability. Note that fast payments lead to mutations in the account balances of the involved money routers. This problem is addressed in Section VI.

Fast payments have three major advantages for users. First, it creates an open ecosystem for settlement activities, which benefits transparency and reduces the need for a central bank. Second, inter-bank settlement durations are significantly decreased, from days to seconds. Third, we reduce costs for inter-bank payments since no communication between banks is required, except when restoring balances (see Section VI).

Our system shares characteristics with services provided by Transferwise. Transferwise is a currency exchange service to offer a cheaper alternative to established institutions when making international payments. It routes payments not by transferring the sender's money directly to the recipient, but by redirecting them to the recipient of an equivalent transfer going in the opposite direction. The essential idea is to convert international money transfers into a sequence of local transactions. Their approach is comparable with our money router mechanism, as it also aims to reduce fees and improve efficiency of traditional payments. However, international payments with Transferwise can still take a few days to complete, depending on the settlement duration of involved banks.

In the remainder of this work, we elaborate our trust model and technical specifications of money routing. This includes an overlay network where any individual is able to quickly route money between bank accounts. A screenshot of our built Android application is shown in Figure 2[1]. The mobile application allows interfacing with different banks and the initiation of real-time payments using our overlay network.

V. Building Trust using Blockchain Constructs

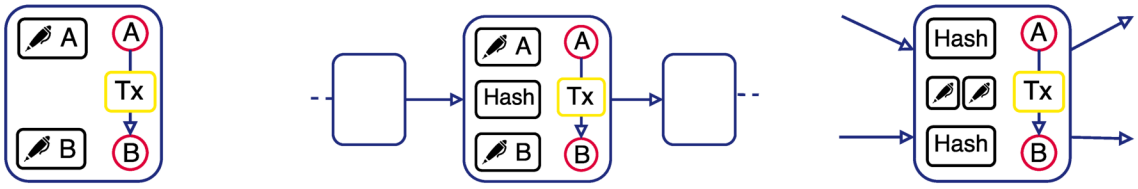
We now explain our deployed, scalable blockchain fabric to gradually build trust between fast payment initiators and money routers: *Trustchain*. Trustchain is designed around transacting entities and is able to accurately capture interactions between users. We have successfully explored usage of TrustChain for bandwidth accounting, attestations and decentralized trading in prior work [16]. For an elaborate evaluation of Trustchain, we refer the reader to our published article [17].

Figure 3 illustrates how a transaction is recorded between two users on Trustchain. Figure 3a shows a single transaction (Tx). Both parties sign the transaction with any cryptographically secure digital signature algorithm (our implementation uses ECDSA). This makes participation irrefutable and acts as an agreement for the transaction specifications. These digital signatures can efficiently be verified by others. After signing, the transaction is committed to the local databases of both transacting users.

A natural way to order records in a database is to chain them together, ordered by creation time. This is shown in Figure 3b where each record is extended with a pointer that points back to the prior record. In particular, this pointer is a hash computed from the description of the prior record using any cryptographically secure hashing algorithm (our implementation uses SHA256). Each record is equipped with a sequence number $s \in \mathbb{Z}$ (the sequence number of the genesis record is 1). This database organisation resembles a blockchain data structure. While cryptocurrencies like Bitcoin and Ethereum operate on a global blockchain, Trustchain gives each user their own personal chain.

Outside for the user operating a chain, the structure shown in Figure 3b is void of any control. Consequentially, a user is able to tamper with his historical transactions. For instance, individuals are able to remove transactions that are not beneficial for their standing in the network. After modification of a record, validity of the chain can simply be restored by recomputing all prior pointers. To protect against local modifications, we extend each record with an additional pointer that points to the prior record in the chain of the transaction counterparty. This ensures that each record has exactly two incoming and two outgoing pointers, as shown in Figure 3c.

[1]Android mobile application:http://www.ds.ewi.tudelft.nl/fileadmin/pds/homepages/vos/iom/iom.apk



(a) A transaction (Tx) between two users (A and B), with two digital signatures.

(b) A blockchain of transactions. Each record in the chain points back to the previous one.

(c) To increase security, each record also references a record in the chain of the other transaction participant.

Fig. 3: Recording a transaction between two users A and B in Trustchain.

When two users transact, their chains essentially become interleaved or "entangled". This property makes fraud impractical to hide since a counterparty is able to proof malicious activities by revealing his record of the disputed transaction. When users initiate more transactions with others, they quickly become entangled in the network, leading to a directed acyclic graph (DAG) structure as shown in Figure 4. This figure shows seven records, created by seven unique participants. Users are able to collect records stored by others. This ensures adequate replication of Trustchain records throughout the network.

Recording Payments: We use the Trustchain data structure to record money transfers between individuals. Each payment and fast payment is assigned a unique identifier. We define two different transaction types:

1) *commit*: This transaction is a public commitment by a money router to forward received funds. It is signed by the initiator of a fast payment and other money routers, prior to transferring any money. The transaction includes the identifier of a fast payment and account address of the money router that should forward funds.
2) *sent*: This transaction type is signed by two parties involved in an in-house payment and implies that money has been sent and received. This transaction includes the fast payment identifier and a boolean that is true if and only if the payment volume is above a threshold t.

We deliberately choose to hide the exact payment volumes due to privacy considerations, at the cost of reduced information.

Detect and Punish Fraud: The most challenging scenario occurs when a money router promises to forward money, but fails to do so. Since Trustchain provides us with a public ledger, disputes can be detected between transacting entities. Consider the situation when a router promised to forward money to Alice and claimed to have done so but Alice has not observed these funds (yet). Other individuals are informed of this situation when they observe a *commit* transaction, signed by both parties, and a *sent* transaction that is only signed by the money router that didn't forward the funds yet. As soon as such a dispute is detected, we do not consider this money router as intermediary for future fast payments until the dispute has been resolved. Note that a dispute can also occur when a router is unable to forward money, due to downtime of involved banks or insufficient account balance.

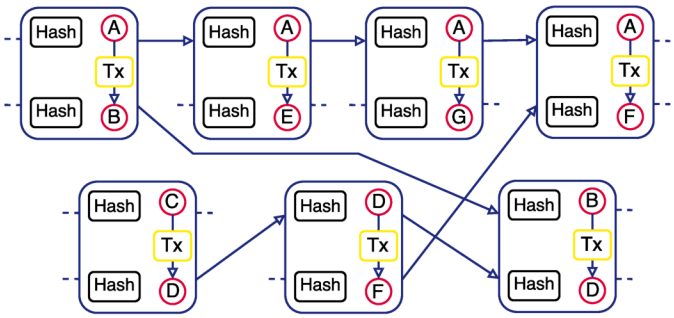


Fig. 4: The tamper-proof Trustchain data structure to record transactions.

In addition to the aforementioned scenario, a user can intentionally lie that he or she has not received funds from a money router. It is impossible to make statements about the status of a specific payment without access to both bank accounts involved in a payment. To resolve disputes, we propose to use input from a dispute arbitrator in the form of an official, digitally signed statement. The dispute arbitrator can be any company that is able to query bank accounts, for instance, the bank involved in a fast payment. A statement provides the status of a fast payment with a specific identifier and should be published on the Trustchain ledger. To discourage users from purposely creating disputes, the arbitrator should charge a small fee for publishing a statement, say €0.10. This fee should be covered by the party that made a false statement about money being sent or received. Note that dispute arbitration enables a new business model for banks.

Quantifying Trustworthiness: We now discuss a mechanism to quantify trustworthiness of honest money routers. Our proposed solution is based on past settlement services provided by money routers. We define a credit network G that models how much money a participant trusts to another individual. The graph is built using collected, dual-signed Trustchain transactions from others. Let $T_{a,b,R}$ indicate a successful money transfer from user a to b using the routers in the set R. Let (a, b, w) indicates a directed edge in G from user a to b with weight w. Now, each identity in our Trustchain network is modelled as a node in G. For each $T_{a,b,R}$ and each router $r \in R$, we create two directed edges: (a, r, w) and (b, r, w) where $w = min(0.01, t)$ (the minimum monetary value we trust to someone is €0.01). These edges represent trust in routers that have forwarded incoming money in the past.

To determine trust scores, we use an algorithm which has been studied extensively in related work, personalised PageRank [18]. The algorithm assigns a score between 0 and 1 to each node in G. These scores are used to pick intermediaries for money forwarding (see Section VI). We consider the node in G that performs the computation as trusted source. Using a reputation algorithm based on random walks is attractive due to its high scalability and low computational complexity. However, one might consider using a reputation algorithm based on maximum network flow to compute trust scores. In particular, we believe the Bazaar algorithm is suitable for this use case and provides additional security at the cost of increased computational requirements [19].

Preventing the Sybil Attack: We propose a mechanism called *router validation* to ensure that a specific bank account can only be operated by a single money router. The effectiveness of this method comes from the difficult and costly process of opening many accounts with different banks internationally. This addresses the challenging Sybil Attack, where an attacker operates multiple entities that use the same bank account for money routing. A router first registers a bank account by sending €0.01 to a trusted third party (TPP), for instance, a bank. The digital identity of TPPs are publicly available. TPPs sign and store a so-called *verify* transaction on Trustchain together with a money router when the payment is observed. This transaction uniquely connects a bank account to a money router. Routers reusing accounts across multiple identities can be identified by querying Trustchain records.

VI. System Design of Internet-of-Money

We expand upon fast payments and our trust model by designing a novel overlay network named *Internet-of-Money*. It operates on top of existing inter-bank payment systems, similar to how The Internet was built on top of the legacy telephone infrastructure.

The Money API: Except for the German FinTS payment protocol, there are no open standards yet for online banking. European legislation called PSD2 is forcing all EU banks to create open interfaces (APIs) [20]. We created one of the first open implementations capable of communicating with numerous banks. We combined banks in the Netherlands (Rabobank, ING and ABN Amro), the British bank HSBC and the Luxembourg payment provider PayPal [21][22]. We devised a single API to communicate with all these banks, called *The Money API*. The Money API provides primitives to login, fetch account balance, query mutations, initiate payments to other accounts and register devices. This library is designed to be extendible and we have partial support for banks in Italy, Greece, Sri Lanka, Turkey and Germany. Our open source[2] library is currently being tested.

Money Routers: Each money routers must offer settlement services with at least two different bank accounts. Having many money routers in the network directly benefits availability and load balancing. A study conducted by NGData indicated that 37.7% of the respondents held accounts at different banks and are able to act as settlement intermediary for money transfers [23]. To create incentives for users to operate a money router, we include transaction fees. Transaction fees can be either fixed, defaulting to €0.01, or a percentage of a fast payment volume. These fees are necessary to cover costs enforced by banks when initiating cross-border payments or when using business accounts to route money. In addition, users can specify a minimum account balance to avoid taking costs when their balance becomes negative. In the remainder of this work, we assume transaction fees are fixed. We also consider an analysis of monetary incentives out of scope and not fundamental for the prototype evaluated in this work.

Note that our design also allows the role of money router to be fulfilled by a single trusted third party or by a few selected trustworthy entities (i.e. financial institutions). A more centralized architecture would mitigate some of the trust and security issues that arise from full decentralization. However, we consider open enrollment (the opportunity for any user to act as a money router) a cardinal property of our system.

Router Discovery: We designed a gossip protocol for discovery of available money routers, based on utility. Like all our proposed infrastructure, it does not depend on any server, company, or other central entity. If Alice wishes to discover a new router, she asks one of her known peers, say Bob, to introduce a router to her. Now, Bob tries to introduce a router to Alice through which she can route money. In general, the algorithm prioritizes routers that provide the most benefit to Alice. If Bob has no router in his set of known peers that are able to provide new services to Alice, he will introduce a random router to Alice. Repeating this gossiping protocol quickly converges to a network with connections between individuals able to provide routing services for each other. An evaluation of this mechanism is given in Section VII-B.

Building a Money Circuit: Prior to transferring money, an initiator of a fast payment starts by selecting eligible routers that are capable of handling the upcoming fast payment. We define a *money circuit* as the set of peers that are involved in a fast payment. This set contains at least one initiator and one beneficiary, and optionally one or more money routers. A money circuit that contains n money routers, is called a n-hop circuit. Building a money circuit proceeds in a depth-first manner and starts with the initiator selecting a router, say r, that is capable of routing money to another account. Next, the initiator sends an *extend* message to r which contains the payment volume and the destination bank account of the fast payment. r responds with a boolean that indicates whether r has sufficient funds to handle the transfer. The response also includes a list of routers that are able to extend the money circuit, and the transaction fee charged by r. If r is able to handle the transfer, the initiator picks a router to extend the circuit with and sends an *extend* message again. These routers are picked based on trustworthiness scores. This process repeats until the initiator built a money circuit that

[2]The Money API source code:
http://www.ds.ewi.tudelft.nl/fileadmin/pds/homepages/vos/iom/internet_of_money.zip

can handle the fast payment. Users are able to change the maximum number of routers in a circuit, which defaults to 3.

The trust model discussed in Section V is based purely on past transactions. It is useful to consider other properties when picking eligible money routers, such as transaction fees, availability, reliability or network latency. Depending on the situation, one might favour low network latency or competitive transaction fees over trustworthiness.

Transferring Money: We now elaborate the process of transferring money over a n-hop circuit. If $n = 0$, money is sent directly to the beneficiary using exactly one in-house payment and no money routers. A single *sent* transaction is created between the fast payment initiator and beneficiary.

When a money circuit involves one or more money routers ($n \geq 1$), the fast payment is facilitated by intermediaries. Let r_i indicate the i-th router in the circuit (r_1 represents the first router). The initiator starts by sending a message to r_1, containing the payment volume and all subsequent routers involved in the money circuit, including the final beneficiary of the fast payment. Next, the initiator initiates a *commit* transaction with r_1 and sends the money. r_1 now starts to poll for the money and finally constructs a *sent* transaction when funds are observed. r_1 forwards the funds to the next router or the beneficiary and this process repeats until the money arrives in the bank account of the beneficiary. The final transfer to the beneficiary does only result in a *sent* transaction. Thus, a fast payment with n intermediaries results in $2n + 1$ new records.

Risk Mitigation: In addition to our trust model, we propose two risk mitigation techniques to reduce counterparty risk when using money routers:

1) *Incremental settlement*: A key risk mitigation technique is to avoid making a single, large payment at once. Instead, a payment is divided into n smaller inter-bank payments. While this increases duration of a fast payment by a factor n, it significantly reduces risk and incentives for intermediaries to compromise money. We believe that reduced risk for some increased latency is a desirable trade-off in Internet-of-Money.
2) *Multi-flow payments*: We uniformly divide a fast payment amongst multiple, distinct money circuits. This results in smaller payments through intermediaries.

While these individual strategies are viable to mitigate counterparty risk, combining them results in a significant reduction of the value at stake, at the cost of additional latency and communication overhead. We evaluate the effectiveness of these strategies in Section VII-B.

Router Recharging: Since funds arrive in one account and leave another, money routers might become insolvent at one point in time, unable to route additional funds. This can be addressed by handling fast payments going in the opposite direction, which restores account balances. However, initiation of these fast payments is outside the control of money routers. Balances can also be restored by initiating a payment from the account with excessive balance to the other bank account. Since this involves an inter-bank payment, settlement might be slow and in turn, this negatively impacts router availability.

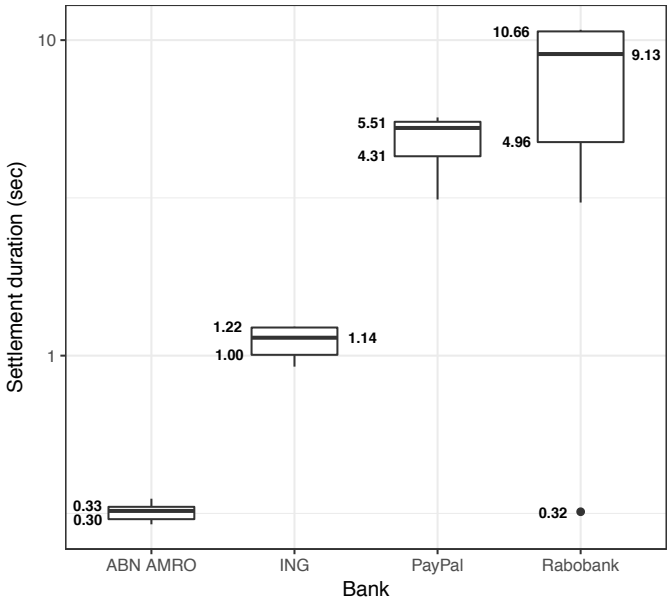


Fig. 5: Settlement durations of in-house payments for four supported banks.

We envision an infrastructure where routers help each other to restore balances, effectively creating a two-sided market with capacity supply and demand. For instance, a router can offer PayPal capacity in return for HSBC funds. While this is an efficient method to restore balances, only requiring in-house payments, we consider the design and implementation of such a mechanism as future work.

VII. Experiments and Evaluation

We now evaluate the performance of money routers, speed of router discovery within Internet-of-Money and the effectiveness of our trust model.

A. Performance of Money Routing

This section concentrates on the performance of fast payments using money routers. All these experiments are conducted with real bank accounts and real money.

Settlement duration of in-house payments: To determine settlement duration of in-house payments for each bank, we send €0.01 ten times between two accounts with different holders, within the same bank. By adding a unique identifier to the description field of a payment, we are able to track payments and accurately measure settlement times. The experiment is executed with two clients on two different computers, with a polling interval of 500 milliseconds, to avoid hammering the bank servers. Polling starts when the payment request has been finished by the sending party. The results are shown in Figure 5, with a non-linear vertical axis. Only one bank, ABN AMRO, has sub-second settlement times with an average duration of 320 milliseconds. ING is slower with 1109 milliseconds on average. PayPal and Rabobank show settlement durations that are an order of magnitude slower, averaging to 4.82 and 7.61 seconds respectively. When performing measurements for the

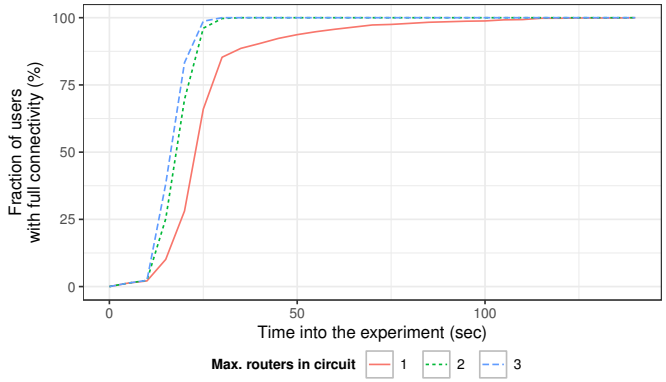


Fig. 6: Performance of router discovery under a varying number of maximum hops in a money circuit.

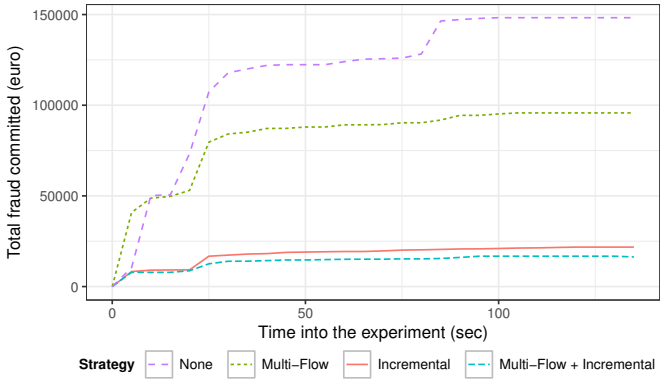


Fig. 7: The effectiveness of fraud prevention, with different risk mitigation strategies.

Rabobank, we observed a notable outlier with a settlement time of 320 milliseconds. This observation can be explained if we assume that similar internal payments might be handled in different ways by the Rabobank. This experiment demonstrates that in-house payments are usually settled within seconds.

International Real-time Money Routing: Next, we focus on the performance of an international fast payment and measure the duration of a money transfer from Rabobank to ABN AMRO, using two money routers. This experiment aims to show the viability and speed of Internet-of-Money. Figure 8 shows the experimental setup and timeline of our experiment.

First, an initiator sends funds from his or her Rabobank account to the first router (holding an account at Rabobank and PayPal), and informs it about the sent funds. Next, the first router starts polling for incoming funds, with an interval of 500 milliseconds. When the first router observes the funds, it forwards them to the second router (holding an account at PayPal and ABN AMRO) and informs this router. When the second router observes the funds, it forwards the money from it's ABN AMRO account to the ABN AMRO account of the beneficiary. In total, three in-house payments are made, with six different bank accounts.

From Figure 8, we conclude that it takes 15.85 seconds in total for money to arrive in the bank account of a beneficiary when using two intermediate routers. A significant amount of time is spent on waiting for the funds to arrive in the PayPal account of the second router, around 6 seconds or 38% of the total duration. The average time to perform a payment is 2.14 seconds and initiation of payments take 41% of the total duration. The average time that a transaction is in transit is 3.02 seconds. The total time to perform a fast payment is heavily influenced by the type and number of intermediate routers. This experiment demonstrates that Internet-of-Money is capable of real-time money routing to other banks.

B. Overlay Evaluation

The purpose of the following experiments is to quantify the performance of our money router overlay. This includes an evaluation of our trust model and effectiveness of fraud detection. We implemented our trust model and Internet-of-Money overlay network in the Python programming language. Our implementation is built upon the Dispersy framework, providing primitives for peer discovery, decentralized communication and secure messaging [24].

Experimental Setup: The following real-world emulations are executed on the DAS-5 supercomputer, using 50 instances per node [25]. We deploy our experiment using the Gumby framework and we create a scenario file where we schedule actions at specific times. All code used during these experiments is open source[3]. Due to the limited number of accounts we own and to avoid a large load on the banking infrastructure, simulated accounts are used during this experiment. We assume a total of five different banks and devised a basic RESTful banking server that handles account creation, payments, balance queries and mutation requests. Distribution of bank accounts amongst users follows the data as published in the NGData customer banking survey (we assume that every user owns at least one bank account) [23].

Router Discovery: We evaluate the efficiency of the router discovery protocol discussed in Section VI. During the experiment, we record the connected peers for each user at a fixed interval (every 5 seconds). We determine whether this user is capable of transferring money to all five different bank accounts, using at most one, two and three intermediate money routers respectively.

Figure 6 shows the performance of router discovery in the Internet-of-Money overlay. The horizontal axis denotes the time into the experiment. The vertical axis indicates the percentage of users that are able to make fast payment to all five banks, or are fully connected. We vary the maximum number of routers in a money circuit. As expected, it takes longer before users are able to build circuits to all other banks using only one router, compared to three routers. However, the differences are marginal. In general, router discovery happens fast: 50% of all users are able to make fast payments to all banks within 25 seconds after the experiment starts. 40 seconds into the experiment, this percentage increased to 90%. Note

[3]https://github.com/devos50/gumby/tree/iom_experiment

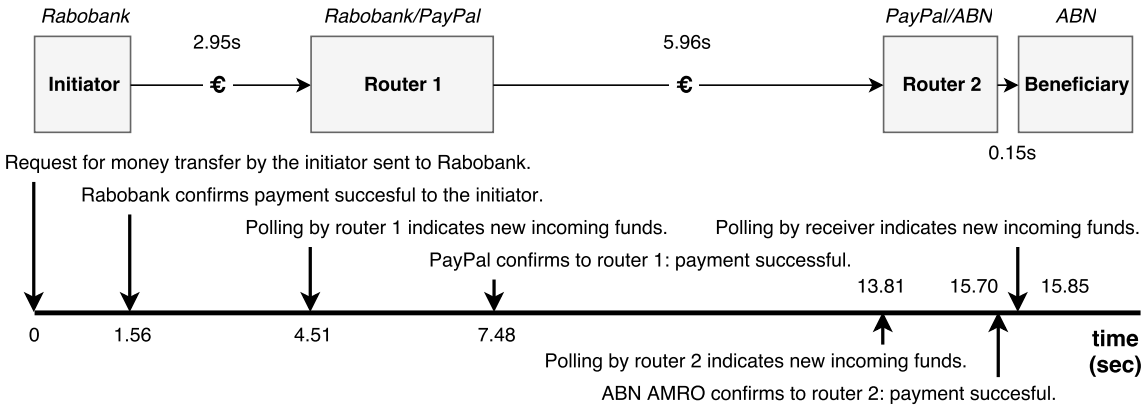


Fig. 8: Timeline of an international fast payment from Rabobank to ABN AMRO, using two money routers.

that it takes longer before *all* users are fully connected using at most one intermediate router: 140 seconds.

Fraud Detection: Our final experiment focusses on the effectiveness of fraud detection (see Section V). To this end, we emulated 200 users with one or more bank accounts. Every five seconds, each user with a single account initiates a fast payment to another entity that has exactly one account of a different type. This forces a money router in the established circuits. The volume of each fast payment is picked from a uniform random distribution between €0.01 and €1000. We challenge ourselves and assume that every user with at least two different bank accounts is malicious and has a 50% probability of committing fraud and not forwarding received funds during a fast payment. To improve router availability, we connect all peers together before the experiment starts. In total, we schedule payments which volume sums to €1,251,848.35.

The results are shown in Figure 7. The horizontal axis denotes the time into the experiment in seconds, after users start performing fast payments to each other. The vertical axis shows the total amount of committed fraud in Euro. We run the experiment four times with different risk mitigation strategies, namely incremental settlement (we split each fast payment in five equal parts) and/or multi-flow payments. The figure hints that the amount of fraud is capped and that malicious routers are successfully excluded from money circuits. Without any risk migration strategy, malicious routers are able to steal €1,544 on average during the whole experiment, indicating that fraudulent routers are able to commit fraud multiple times. This can be addressed to the fact that they are included in multiple money circuits roughly at the same time. If we consider risk mitigation strategies, we see that the combination of multi-flow payments and incremental settlement leads to the lowest amount of fraud possible, on average €174. Using exclusively incremental settlement leads to a slightly higher amount of fraud.

VIII. Discussion

We now discuss this research from various perspectives.

Legal: The idea of directly sharing funds with others, without a central bank involved, challenges existing regulation. Routing money through other bank accounts resembles activity performed by financial settlement institutions and might require a legal prerequisite in the form of a banking license. The PSD2 regulation states that trusted third parties (TPPs) can be authorized by end-users to perform financial activities on their behalf [20]. However, it is unclear whether the definition of a TPP includes money routers. Another consideration is responsibility when a mistaken payment is initiated. Finally, compatibility of our system with (inter)national anti-money laundry regulations is uncertain. Exploring legal compliance of this work is a fundamental requirement for broader adoption.

Limitations: While we have proven the viability of our idea, there are several limitations that must be addressed prior to broader adoption. We noticed that banks are not used to our dynamic way of initiating money transfers and our accounts got blocked several times due to suspected fraudulent behaviour. An open ecosystem for settlement demands changes by banks and it is an open question whether they are willing to do so. On the other hand, many banks are already forced to innovate their legacy systems to remain competitive [15].

Additionally, we observed that some banks require two-factor authentication when transferring funds to unknown bank accounts. This limits automation of money transfers since a manual action by the user is required for a payment to proceed.

Privacy: We consider privacy an important requirement of our open platform and expose minimal information about money flows. The current privacy model in Internet-of-Money is effective but open for extension. Decentralized path-based transaction networks, for instance, SpeedyMurmurs, aim to solve this specific problem [26].

Scalability: Our overlay network is scalable, due to the absence of global consensus. However, techniques like incremental settlement lead to additional payments and a higher load on the banks. In addition, the choice of reputation mechanism used in Internet-of-Money influences scalability.

IX. Related Work

The last few years, there has been a steep increase in Fintech start-ups, eager to disrupt existing financial services. Hawala is an informal system to transfer value, without actually moving

money [27]. It consists of a network of hawala brokers, that take a small commission. In contrast to our system, trust in hawala is cultivated in an analogue manner whereas our model depends on a digital solution.

Innovation in the financial sector has been catalysed by the popularity of Blockchain technology, aiming to build trust between strangers without involvement of centralized authorities. Bitcoin has proven that a sustainable currency can be built without a central bank in control [2]. However, wide-spread adoption stays out due to its volatile pricing, high transaction fees, relatively slow confirmation times and unsure future. The Lightning Network aims to improve scalability of Bitcoin by providing bi-directional payment channels between users [28]. Payments between two users not directly connected with a payment channel, are realised by routing payments through channels of other users. This has similarities with money routing in Internet-of-Money. New usages of blockchain technology are focussed around the way users transfer money and other assets. The Ripple project, supported by various major banks, attempts to build a connected network of financial institutions and payment providers [29]. Their solution aims to significantly speed up traditional money transfers, lower costs and provide support for high-volume transactions. R3 Corda can be compared to Trustchain since they share the idea that a ledger with global consistency is often not necessary [30].

While blockchain solutions are slowly being adopted, the aforementioned systems all aim to increase utility by building a financial network from scratch. In comparison, Internet-of-Money is built upon existing, proven infrastructure, making migration towards our system effortless.

X. Conclusions and Future Work

We explored a new stage in the evolution of digital trust and addressed the problem of trusting strangers with your money. The tamper-proof Trustchain structure provides a scalable and public trace of historical interactions, and allows detection and punishment of potential fraud. We expand upon this with an overlay network to transfer money within seconds to others, using other network participants as financial intermediaries. This mechanism depends on the fast settlement of in-house payments. Our open ecosystem dramatically improves speed when initiating cross-border payments while preserving privacy and scalability. Our experiments demonstrated the efficiency of in-house payments and effectiveness of money routers. Additionally, we have proven that our fraud detection mechanism, together with incremental settlement and multi-flow payments, limits misuse and punishes malicious behaviour. However, there are various legal issues and limitations that should be addressed, mostly by financial institutions, before broader usage can be realised.

This work is an important milestone in our ambitious vision to create the *programmable economy*. Ongoing work towards this goal addresses self-sovereign identity, scalable blockchain consensus compatible with Trustchain, and decentralized marketplaces. We refer the interested reader to our scientific overview article [16].

Acknowledgements

The authors would like to thank Laurens Versluis for his initial contributions to the design and implementation of The Internet-of-Money.

References

[1] P. Resnick *et al.*, "Trust among strangers in internet transactions: Empirical analysis of ebays reputation system," *The Economics of the Internet and E-commerce*, vol. 11, no. 2, pp. 23–25, 2002.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] T. Kokkola, *The payment system: Payments, securities and derivatives, and the role of the Eurosystem*. European Central Bank, 2011.

[4] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[5] R. McMillan, "The inside story of mt. gox, bitcoins 460 million disaster," *Wired. March*, vol. 3, 2014.

[6] "The uber data breach has implications for us all." [Online]. Available: https://www.ft.com/content/e2bf6caa-d2cb-11e7-a303-9060cb1e5f44

[7] Z. Yan and S. Holtmanns, "Trust modeling and management: from social trust to digital trust," *IGI Global*, pp. 290–323, 2008.

[8] D. M. Kreps *et al.*, "Rational cooperation in the finitely repeated prisoners' dilemma," *Journal of Economic theory*, vol. 27, no. 2, pp. 245–252, 1982.

[9] B.-J. Koops, "The trouble with european data protection law," *International Data Privacy Law*, vol. 4, no. 4, pp. 250–261, 2014.

[10] R. Delaviz *et al.*, "Sybilres: A sybil-resilient flow-based decentralized reputation mechanism," in *ICDCS, 2012*. IEEE, 2012, pp. 203–213.

[11] S. D. Kamvar *et al.*, "The eigentrust algorithm for reputation management in p2p networks," in *WWW*. ACM, 2003, pp. 640–651.

[12] M. Srivatsa *et al.*, "Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of WWW'05*. ACM, 2005, pp. 422–431.

[13] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 251–260.

[14] *SWIFT payment system*. [Online]. Available: https://www.swift.com

[15] McKinsey, *Global Payments 2016*, 2016 (accessed November 27, 2017).

[16] J. Pouwelse *et al.*, "Laws for creating trust in the blockchain age," *European Property Law Journal*, 2017.

[17] P. Otte *et al.*, "Trustchain: A sybil-resistant scalable blockchain," *Future Generation Computer Systems*, 2017.

[18] L. Page *et al.*, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[19] A. Post *et al.*, "Bazaar: Strengthening user reputations in online marketplaces," in *Proceedings of NSDI11*, 2011, p. 183.

[20] M. Cortet *et al.*, "Psd2: The digital transformation accelerator for banks," *Journal of Payments Strategy & Systems*, vol. 10, no. 1, pp. 13–27, 2016.

[21] J. Doe and J. Pouwelse, "A vulnerability analyis of smartphone banking applications," 2015, unpublished research.

[22] J. Awesome and J. Pouwelse, "A vulnerability analysis of mobile banking applications," 2015, unpublished research.

[23] N. n, *NGDATA 2014 Consumer Banking Survey*, 2014 (accessed November 20, 2017). [Online]. Available: http://www.ngdata.com/wp-content/uploads/NGDATA-2014-consumer-banking-survey-brief.pdf

[24] N. Zeilemaker, B. Schoon, and J. Pouwelse, "Dispersy bundle synchronization," *TU Delft, Parallel and Distributed Systems*, 2013.

[25] H. Bal *et al.*, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, no. 5, pp. 54–63, 2016.

[26] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," *arXiv preprint arXiv:1709.05748*, 2017.

[27] P. M. Jost and H. S. Sandhu, *The hawala alternative remittance system and its role in money laundering*. Interpol, 2003.

[28] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments. 2016," 2015.

[29] D. Schwartz *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, 2014.

[30] R. G. Brown, "Introducing r3 corda: A distributed ledger for financial services," *R3, April*, vol. 5, 2016.

Virtual Network Embedding Approximations: Leveraging Randomized Rounding

Matthias Rost
TU Berlin, Germany
Email: mrost@inet.tu-berlin.de

Stefan Schmid
University of Vienna, Austria
Email: stefan_schmid@univie.ac.at

***Abstract*—The Virtual Network Embedding Problem (VNEP) captures the essence of many resource allocation problems of today's infrastructure providers, which offer their physical computation and networking resources to customers. Customers request resources in the form of Virtual Networks, i.e. as a directed graph which specifies computational requirements at the nodes and communication requirements on the edges. An embedding of a Virtual Network on the shared physical infrastructure is the joint mapping of (virtual) nodes to physical servers together with the mapping of (virtual) edges onto paths in the physical network connecting the respective servers.**

This work initiates the study of approximation algorithms for the VNEP. Concretely, we study the offline setting with admission control: given multiple request graphs the task is to embed the most profitable subset while not exceeding resource capacities. Our approximation is based on the randomized rounding of Linear Programming (LP) solutions. Interestingly, we uncover that the standard LP formulation exhibits an inherent structural deficit when considering general virtual networks: its solutions cannot be decomposed into valid embeddings. In turn, focusing on the class of cactus request graphs, we devise a novel LP formulation, whose solutions can be decomposed into convex combinations of valid embedding. Proving performance guarantees of our rounding scheme, we obtain the first approximation algorithm for the VNEP in the resource augmentation model.

We propose two rounding heuristics and evaluate their performance in an extensive computational study, showing that these consistently yield good solutions (even without augmentations).

I. Introduction

Cloud applications usually consist of multiple distributed components (e.g., virtual machines, containers), which results in substantial communication requirements. If the provider fails to ensure that these communication requirements are met, the performance can suffer dramatically [1]. Consequently, over the last years, several proposals have been made to *jointly* provision the computational functionality *together* with appropriate network resources. The Virtual Network Embedding Problem (VNEP) captures the core of this problem: given a directed graph specifying computational requirements at the nodes and bandwidth requirements on the edges, an embedding of this *Virtual Network* in the physical network has to be found, such that both the computational and the network requirements are met. Figure 1 illustrates two incarnations of virtual networks: *service chains* [2] and *virtual clusters* [3].

We study the offline setting with admission control: given multiple requests the task is to embed the most profitable subset while not exceeding resource capacities.

ISBN 978-3-903176-08-9

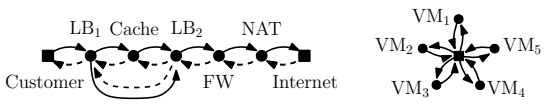


Fig. 1. Examples for virtual networks 'in the wild'. The left graph shows a service chain for mobile operators [4]: load-balancers route (parts of the) traffic through a cache. Furthermore, a firewall and a network-address translation are used. The right graph depicts the Virtual Cluster abstraction for provisioning virtual machines (VMs) in data centers. The abstraction provides connectivity guarantees via a *logical switch* in the center [3].

A. Formal Problem Statement

In the light of the recent interest in Service Chaining [2], we extend the VNEP's general definition [5] by considering different *types* of computational nodes. We refer to the physical network as the *substrate network*. The substrate $G_S = (V_S, E_S)$ is offering a set $\mathcal{T}$ of computational types. This set of types may contain e.g., 'FW' (firewall), 'x86 server', etc. For a type $\tau \in \mathcal{T}$, the set $V_S^\tau \subseteq V_S$ denotes the substrate nodes that can host functionality of type τ. Denoting the node resources by $R_S^V = \{(\tau, u) \mid \tau \in \mathcal{T}, u \in V_S^\tau\}$ and all substrate resources by $R_S = R_S^V \cup E_S$, the capacity of nodes and edges is denoted by $d_S(x, y) > 0$ for $(x, y) \in R_S$.

For each request $r \in \mathcal{R}$, a directed graph $G_r = (V_r, E_r)$ together with a profit b_r is given. We refer to the respective nodes as virtual or request nodes and similarly refer to the respective edges as virtual or request edges. The types of virtual nodes are indicated by the function $\tau_r : V_r \to \mathcal{T}$.

Based on policies of the customer or the provider, the mapping of virtual node $i \in V_r$ is restricted to a set $V_S^{r,i} \subseteq V_S^{\tau_r(i)}$, while the mapping of virtual edge (i, j) is restricted to $E_S^{r,i,j} \subseteq E_S$. Each virtual node $i \in V_r$ and each edge $(i, j) \in E_r$ is attributed with a resource demand $d_r(i) \geq 0$ and $d_r(i, j) \geq 0$, respectively. Virtual nodes and edges can only be mapped on substrate nodes and edges of sufficient capacity, i.e. $V_S^{r,i} \subseteq \{u \in V_S^{\tau_r(i)} | d_S(u) \geq d_r(i)\}$ and $E_S^{r,i,j} \subseteq \{(u, v) \in E_S | d_S(u, v) \geq d_r(i, j)\}$ holds.

We denote by $d_{\max}(r, x, y)$ the maximal demand that a request r may impose on a resource $(x, y) \in R_S$:

$$d_{\max}(r, \tau, u) = \max(\{0\} \cup \{d_r(i) | i \in V_r : \tau(i) = \tau \wedge u \in V_S^{r,i}\})$$
$$d_{\max}(r, u, v) = \max(\{0\} \cup \{d_r(i, j) | (i, j) \in E_r : (u, v) \in E_S^{r,i,j}\})$$

In the following the notions of valid mappings (respecting mapping constraints) and feasible embeddings (respecting resource constraints) are introduced to formalize the VNEP.

Definition 1 (Valid Mapping)**.** *A valid mapping m_r of request $r \in \mathcal{R}$ is a tuple (m_r^V, m_r^E) of functions $m_r^V : V_r \to V_S$ and $m_r^E : E_r \to \mathcal{P}(E_S)$, such that the following holds:*

- *Virtual nodes are mapped to allowed substrate nodes: $m_r^V(i) \in V_S^{r,i}$ holds for all $i \in V_r$.*
- *The mapping $m_r^E(i,j)$ of virtual edge $(i,j) \in E_r$ is an edge-path connecting $m_r^V(i)$ to $m_r^V(j)$ only using allowed edges, i.e. $m_r^E(i,j) \subseteq \mathcal{P}(E_S^{r,i,j})$ holds.*

We denote by $\mathcal{M}_r$ the set of valid mappings of request $r \in \mathcal{R}$.

Definition 2 (Allocations of Valid Mappings)**.** *We denote by $A(m_r, x, y)$ the cumulative allocation induced by the valid mapping $m_r \in \mathcal{M}_r$ on resource $(x,y) \in R_S$:*

$$A(m_r, \tau, u) = \sum\nolimits_{i \in V_r, \tau(i)=\tau, m_r^V(i)=u} d_r(i) \qquad \forall (\tau, u) \in R_S^V$$

$$A(m_r, u, v) = \sum\nolimits_{(i,j) \in E_r, (u,v) \in m_r^E(i,j)} d_r(i,j) \ \forall (u,v) \in E_S$$

The maximal allocation that a valid mapping of request $r \in \mathcal{R}$ may impose on a substrate resource $(x,y) \in R_S$ is denoted by $A_{\max}(r,x,y) = \max_{m_r \in \mathcal{M}_r} A(m_r, x, y)$.

Definition 3 (Feasible Embedding)**.** *A feasible embedding of a subset of requests $\mathcal{R}' \subseteq \mathcal{R}$ is a collection of valid mappings $\{m_r\}_{r \in \mathcal{R}'}$, such that the* cumulative *allocations on nodes and edges does not exceed the substrate capacities, i.e. $\sum_{r \in \mathcal{R}'} A(m_r, x, y) \leq d_S(x,y)$ holds for $(x,y) \in R_S$.*

Definition 4 (Virtual Network Embedding Problem)**.** *The VNEP asks for a feasible embedding $\{m_r\}_{r \in \mathcal{R}'}$ of a subset of requests $\mathcal{R}' \subseteq \mathcal{R}$ maximizing the profit $\sum_{r \in \mathcal{R}'} b_r$.*

B. Related Work

In the last decade, the VNEP has attracted much attention due to its many applications and the survey [5] from 2013 already lists more than 80 different algorithms for its many variations [5]. The VNEP is known to be $\mathcal{NP}$-hard and inapproximable in general (unless $\mathcal{P} = \mathcal{NP}$) [6]. Based on the hardness of the VNEP, most works consider heuristics without any performance guarantee [5], [7]. Other works proposed exact methods as integer or constraint programming, coming at the cost of an exponential runtime [8], [9], [10].

A column generation approach was proposed by Jarray et al. in [9] to efficiently compute solutions to the VNEP by generating valid mappings 'on-the-fly'. We believe that our decomposable LP formulations may be used to price (i.e. generate) further valid mappings more efficiently than by using Mixed-Integer Programming.

Acknowledging the hardness of the general VNEP and the diversity of applications, several subproblems of the VNEP have been studied recently by considering restricted graph classes for the virtual networks and the substrate graph. For example, virtual clusters with uniform demands are studied in [11], [3], line requests are studied in [12], [13], [14] and tree requests were studied in [15], [13].

Considering approximation algorithms, Even et al. employed randomized rounding in [13] to obtain a constant approximation for embedding line requests on arbitrary substrate graphs under strong assumptions on *both* the benefits and the capacities. In their interesting work, Bansal et al. [15] give an $n^{O(d)}$ time $O(d^2 \log(nd))$-approximation algorithm for minimizing the load of embedding d-depth trees based on a strong LP relaxation inspired by the Sherali-Adams hierarchy. To the best of our knowledge, no approximation algorithms are known for arbitrary substrate graphs and classes of virtual networks containing cyclic substructures.

Bibliographic Note: In our preliminary technical report [16] similar results were presented. The current work presents a significantly simpler LP formulation and also provides an extensive computational evaluation. An extended version of this work, containing all proofs and additional details on our evaluation, can be found at [17].

Additionally, in our recent technical report [18], the approximation approach presented in this work is extended beyond cactus request graphs. However, approximating more general request graphs comes at the price of non-polynomial runtimes.

C. Outline of Randomized Rounding for the VNEP

We shortly revisit the concept of randomized rounding. Given an Integer Program for a certain problem, randomized rounding works by (i) computing a solution to its Linear Program relaxation, (ii) decomposing this solution into convex combinations of *elementary* solutions, and (iii) probabilistically selecting elementary solutions based on their weight.

Accordingly, for applying randomized rounding for the VNEP, a convex combination of valid mappings $\mathcal{D}_r = \{(f_r^k, m_r^k) | m_r^k \in \mathcal{M}_r, f_r^k > 0\}$ must be recovered from the Linear Programming solution for each request $r \in \mathcal{R}$, such that (i) the profit of these convex combinations equals the profit achieved by the Linear Program and (ii) the (fractional) cumulative allocations do not violate substrate capacities. To round a solution, for each request r the mapping m_r^k is selected with probability f_r^k, rejecting r with probability $1 - \sum_k f_r^k$.

D. Results and Organization

This paper initiates the study of approximation algorithms for the VNEP on general substrates *and* general virtual networks. Specifically, we employ randomized rounding to obtain the first approximation algorithm for the non-trivial class of cactus graph requests in the resource augmentation model.

Studying the classic multi-commodity flow (MCF) formulation for the VNEP in Section II, we show that its solutions can only be decomposed for tree requests: request graphs containing cycles can in general not be decomposed into valid mappings. This result has ramifications beyond the inability to apply randomized rounding: we prove that the MCF formulation exhibits an unbounded integrality gap. Investigating the root cause for this surprising result, we devise a novel *decomposable* Linear Programming formulation in Section III for the class of cactus graph requests. We then present and prove performance guarantees for our randomized rounding algorithm in Section IV, obtaining the first approximation algorithm for the Virtual Network Embedding Problem. Section V presents a synthetic computational study, in which

two rounding heuristics are evaluated. Our results indicate that high-quality solutions can be obtained even without resource augmentations. In particular, our heuristical rounding algorithm achieved 73.8% of the baseline's profit on average.

II. The Classic Multi-Commodity Formulation for the VNEP and its Limitations

In this section, we study the relaxation of the standard multi-commodity flow (MCF) formulation for the VNEP (cf. [2], [7]). We first show the positive result that the formulation is sufficiently strong to decompose virtual networks being *trees* into convex combinations of valid mappings. Subsequently, we show that the formulation fails to allow for the decomposition of *cyclic requests*. This not only impacts its applicability for randomized rounding but renders the formulation useless for approximations in general: it can be shown that the formulation's integrality gap is unbounded (cf. [17]).

A. The Classic Multi-Commodity Formulation

The classic MCF formulation for the VNEP is presented as Formulation 1 . We first describe its integer variant, which computes a single valid mapping for each request by using binary variables. The Linear Programming variant is obtained by relaxing the binary variables' domain to $[0,1]$.

The variable $x_r \in \{0,1\}$ indicates whether request $r \in \mathcal{R}$ is embedded or not. The variable $y^u_{r,i} \in \{0,1\}$ indicates whether virtual node $i \in V_r$ was mapped on substrate node $u \in V_S$. Similarly, the flow variable $z^{u,v}_{r,i,j} \in \{0,1\}$ indicates whether the substrate edge $(u,v) \in E_S$ is used to realize the virtual edge $(i,j) \in E_r$. The variable $a^{x,y}_r \geq 0$ denotes the cumulative allocations of request $r \in \mathcal{R}$ induced on resource $(x,y) \in R_S$.

By Constraint 2, the virtual node $i \in V_r$ of request $r \in \mathcal{R}$ must be placed on any of the suitable substrate nodes in $V_S^{r,i}$ iff. $x_r = 1$ holds and Constraint 3 forbids the mapping on nodes which may not host node i. Constraint 4 induces an unsplittable unit flow for each virtual edge $(i,j) \in E_r$ from the substrate location to which i was mapped to the substrate location to which j was mapped. By Constraint 5 virtual edges may only be mapped on *allowed* substrate edges. Constraints 6 and 7 compute the cumulative allocations and Constraint 8 guarantees that the substrate resource capacities are respected. The following lemma states the connectivity property enforced by Formulation 1 (see [17] for the proof).

Formulation 1: Classic MCF Formulation for the VNEP

$$\max \sum_{r \in \mathcal{R}} b_r x_r \tag{1}$$

$$\sum_{u \in V_S^{r,i}} y^u_{r,i} = x_r \qquad \forall r \in \mathcal{R}, i \in V_r \tag{2}$$

$$\sum_{u \in V_S \setminus V_S^{r,i}} y^u_{r,i} = 0 \qquad \forall r \in \mathcal{R}, i \in V_r \tag{3}$$

$$\begin{bmatrix} \sum_{(u,v)\in\delta^+(u)} z^{u,v}_{r,i,j} \\ -\sum_{(v,u)\in\delta^-(u)} z^{v,u}_{r,i,j} \end{bmatrix} = \begin{bmatrix} y^u_{r,i} \\ -y^u_{r,j} \end{bmatrix} \forall \begin{bmatrix} r \in \mathcal{R}, (i,j) \in E_r, \\ u \in V_S \end{bmatrix} \tag{4}$$

$$z^{u,v}_{r,i,j} = 0 \qquad \forall \begin{bmatrix} r \in \mathcal{R}, (i,j) \in E_r, \\ (u,v) \in E_S \setminus E_S^{r,i,j} \end{bmatrix} \tag{5}$$

$$\sum_{i \in V_r, \tau_r(i) = \tau} d_r(i) \cdot y^u_{r,i} = a^{\tau,u}_r \qquad \forall r \in \mathcal{R}, (\tau,u) \in R_S^V \tag{6}$$

$$\sum_{(i,j)\in E_r} d_r(i,j) \cdot z^{u,v}_{r,i,j} = a^{u,v}_r \qquad \forall r \in \mathcal{R}, (u,v) \in E_S \tag{7}$$

$$\sum_{r \in \mathcal{R}} a^{x,y}_r \leq d_S(x,y) \; \forall (x,y) \in R_S \tag{8}$$

Lemma 5 (Local Connectivity Property of Formulation 1)**.** *For any virtual edge $(i,j) \in E_r$ and any substrate node $u \in V_S^{r,i}$ with $y^u_{r,i} > 0$, there exists a path $P^{u,v}_{r,i,j}$ in G_S from u to $v \in V_S^{r,j}$ with $y^v_{r,j} > 0$, such that the flow along any edge of $P^{u,v}_{r,i,j}$ with respect to the variables $z^{\cdot,\cdot}_{r,i,j}$ is greater 0.*

The path $P^{u,v}_{r,i,j}$ can be computed in polynomial time.

B. Decomposing Solutions for Tree Requests

Given Lemma 5, we now present Algorithm 1 to decompose solutions to the LP Formulation 1 into convex combinations of valid mappings $\mathcal{D}_r = \{(f^k_r, m^k_r) | m^k_r \in \mathcal{M}_r, f^k_r > 0\}$ (cf. Section I-C), *if* the request's underlying undirected graph is a *tree*. Recall that in the LP formulation the binary variables are relaxed to take any value in the interval $[0,1]$.

Given a request $r \in \mathcal{R}$, the algorithm processes all virtual edges according to an arbitrary acyclic representation $G^{\mathcal{A}}_r = (V_r, E^{\mathcal{A}}_r, r_r)$ of the undirected interpretation of G_r being rooted at $r_r \in V_r$. Concretely, the edge set $E^{\mathcal{A}}_r$ is obtained from E_r by reorienting (some of the) edges, such that any node $i \in V_r$ can be reached from r_r. Considering tree requests for now, $G^{\mathcal{A}}_r$ is an arborescence and can be computed by a simple graph search of the underlying undirected graph starting at r_r. We denote by $\overleftarrow{E}^{\mathcal{A}}_r = E_r \setminus E^{\mathcal{A}}_r$ the edges whose orientations were reversed in the process of computing $G^{\mathcal{A}}_r$.

The algorithm extracts mappings m^k_r of value f^k_r iteratively, as long as $x_r > 0$ holds. Initially, in the k-th iteration, none of the virtual nodes and edges are mapped. As $x_r > 0$ holds, there must exist a node $u \in V_S^{r,r_r}$ with $y^{r_r}_{r,i} > 0$ by Constraint 2 and the algorithm accordingly sets $m^V_r(r_r) = u$. Given this initial fixing, the algorithm iteratively extracts nodes from the queue $\mathcal{Q}$ which have been already mapped and considers all outgoing virtual edges $(i,j) \in E^{\mathcal{A}}_r$. If an outgoing edge (i,j) is contained in E_r, Lemma 5 can be readily applied to obtain a joint mapping of the edge (i,j) and its head j. If the edge's orientation was reversed, i.e. if $(i,j) \in \overleftarrow{E}^{\mathcal{A}}_r$ holds, Lemma 5 is applied *while reversing* the flow's direction (see Lines 13-16).

First, note that by the repeated application of Lemma 5, the mapping of virtual nodes and edges is valid. As $G^{\mathcal{A}}_r$ is an arborescence, each edge and each node of $G^{\mathcal{A}}_r$ will eventually be mapped and hence m^k_r is a valid mapping. The mapping value f^k_r is computed as the minimum of the mapping variables $\mathcal{V}_k$ used for constructing m^k_r. Reducing the values of the mapping variables together with the allocation variables $\vec{a}_r$ (Lines 20-21), the Constraints 2-7 continue to hold.

As the decomposition process continues as long as $x_r > 0$ holds and in the k-th step at least one variable's value is set to 0, the algorithm terminates with a complete decomposition

Algorithm 1: Decompositioning MCF solutions for Tree Requests

Input : Tree request $r \in \mathcal{R}$, solution $(x_r, \vec{y}_r, \vec{z}_r, \vec{a}_r)$ to LP Formulation 1, acyclic reorientation $G_r^{\mathcal{A}} = (V_r, E_r^{\mathcal{A}}, r_r)$

Output: Convex combination $\mathcal{D}_r = \{D_r^k = (f_r^k, m_r^k)\}_k$

set $\mathcal{D}_r \leftarrow \emptyset$ **and** $k \leftarrow 1$
while $x_r > 0$ **do**
  set $m_r^k = (m_r^V, m_r^E) \leftarrow (\emptyset, \emptyset)$
  set $\mathcal{Q} = \{r_r\}$
  choose $u \in V_S^{r,r_r}$ **with** $y^u_{r,r_r} > 0$ **and set** $m_r^V(r_r) \leftarrow u$
  while $|\mathcal{Q}| > 0$ **do**
    choose $i \in \mathcal{Q}$ **and set** $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{i\}$
    foreach $(i,j) \in E_r^{\mathcal{A}}$ **do**
      if $(i,j) \in E_r$ **then**
        compute $\overrightarrow{P}^{u,v}_{r,i,j}$ connecting $m_r^V(i) = u$ to $v \in V_S^{r,j}$ according to Lemma 5
        set $m_r^V(j) = v$ **and** $m_r^E(i.j) = \overrightarrow{P}^{u,v}_{r,i,j}$
      else
        let $\overleftarrow{z}^{v',u'}_{r,i,j} \triangleq z^{u',v'}_{r,j,i}$ **for all** $(u',v') \in E_S$
        compute $\overleftarrow{P}^{v,u}_{r,i,j}$ connecting $m_r^V(i) = v$ to $u \in V_S^{r,j}$ according to Lemma 5
        set $\overrightarrow{P}^{u,v}_{r,j,i} = \texttt{reverse}(\overleftarrow{P}^{v,u}_{r,i,j})$
        set $m_r^V(i) = u$ **and** $m_r^E(j,i) = \overrightarrow{P}^{u,v}_{r,j,i}$
      set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{j\}$
  set $\mathcal{V}_k \leftarrow \begin{pmatrix} \{x_r\} \cup \{y^{m_r^V(i)}_{r,i} | i \in V_r\} \\ \cup \{z^{u,v}_{r,i,j} | (i,j) \in E_r, (u,v) \in m_r^E(i,j)\} \end{pmatrix}$
  set $f_r^k \leftarrow \min \mathcal{V}_k$
  set $v \leftarrow v - f_r^k$ **for all** $v \in \mathcal{V}_k$
  set $a_r^{x,y} \leftarrow a_r^{x,y} - f_r^k \cdot A(m_r^k, x, y)$ **for all** $(x,y) \in R_S$
  add $D_r^k = (f_r^k, m_r^k)$ to $\mathcal{D}_r$ **and set** $k \leftarrow k+1$
return $\mathcal{D}_r$

for which $\sum_k f_r^k = x_r$ holds. Furthermore, the algorithm has polynomial runtime, as in each iteration at least one variable is set to 0 and the number of variables for request r is bounded by $\mathcal{O}(|E_r| \cdot |E_S|)$. Hence, we obtain the following:

Lemma 6. *Given a virtual network request $r \in \mathcal{R}$, whose underlying undirected graph is a tree, Algorithm 1 decomposes a solution $(x_r, \vec{y}_r, \vec{z}_r, \vec{a}_r)$ to the LP Formulation 1 into valid mappings $\mathcal{D}_r = \{(m_r^k, f_r^k)\}_k$, such that the following holds:*

- *The decomposition is complete, i.e. $x_r = \sum_k f_r^k$ holds.*
- *The decomposition's resource allocations are bounded by $\vec{a}_r$: $a_r^{x,y} \geq \sum_k f_r^k \cdot A(m_r^k, x, y)$ holds for $(x,y) \in R_S$.*

C. Limitations of the Classic MCF Formulation

Above it was shown that LP solutions to the classic MCF formulation can be decomposed into convex combinations of valid mappings *if* the underlying graph is a tree. This does not hold anymore when considering cyclic virtual networks:

Theorem 7. *Solutions to the standard LP Formulation 1 can in general not be decomposed into convex combinations of valid mappings if the virtual networks contain cycles.*

Proof. In Figure 2 we visually depict an example of a solution to the LP Formulation 1 from which *not a single* valid mapping can be extracted. The validity of the depicted solution follows from the fact that the virtual node mappings sum to 1 and

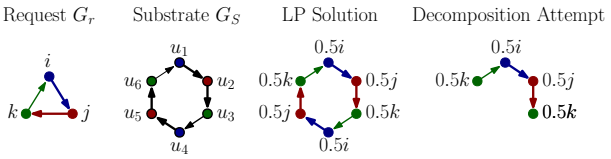


Fig. 2. Example showing that solutions to the LP Formulation 1 can in general not be decomposed into convex combinations of valid mappings. Request r is a simple cyclic graph which shall be mapped on the substrate graph G_S. We assume following node mapping restrictions $V_S^{r,i} = \{u_1, u_4\}$, $V_S^{r,j} = \{u_2, u_5\}$, $V_S^{r,k} = \{u_3, u_6\}$. The LP solution with $x_r = 1$ is depicted as follows. Substrate nodes are annotated with the mapping of virtual nodes. Hence, $0.5i$ at node u_1 indicates $y^{u_1}_{r,i} = 1/2$, i.e. that virtual node i is mapped with 0.5 on substrate node u_1. Substrate edges are colored according to the color of virtual links mapped onto it. Virtual links are all mapped using flow values $1/2$. Accordingly, for example $z^{u_1,u_2}_{r,i,j} = 1/2$ holds.

each virtual node connects to its neighboring node with half a unit of flow. Assume for the sake of contradiction that the depicted solution can be decomposed. As virtual node $i \in V_r$ is mapped onto substrate node $u_1 \in V_S$, and $u_2 \in V_S$ is the only neighboring node with respect to variables $z_{r,i,j}$ that hosts $j \in V_r$, there must exist a mapping (m_r^V, m_r^E) with $m_r^V(i) = u_1$ and $m_r^V(j) = u_2$. Similarly, $m_r^V(k) = u_3$ must hold. However, for $m_r^V(i) = u_1$, the virtual node k must be mapped to u_6, as otherwise the embedding of (k,i) cannot lead to substrate node u_1. Hence the virtual node $k \in V_r$ must be mapped both on u_6 and u_3. As this is not possible, and the same argument holds when considering the mapping of i onto u_4, no valid mapping can be extracted. □

This non-decomposability also induces large integrality gaps, as proven in our extended technical report [17].

Theorem 8. *The integrality gap of the MCF formulation is unbounded. This even holds under infinite substrate capacities.*

III. Novel Decomposable LP Formulation

In this section, we present a novel LP formulation and its accompanying decomposition algorithm for the class of cactus request graphs, i.e. graphs for which cycles intersect in at most a single node (in its undirected interpretation). Accordingly, these graphs can be uniquely decomposed into cycles and a single forest (cf. Lemma 9 below).

Before delving into the details of our novel LP formulation, we discuss our main insight on how to overcome the limitations of the MCF formulation and accordingly how to derive decomposable formulations. To this end, it is instructive, to revisit the non-decomposable example of Figure 2 by applying the decomposition Algorithm 1 on the depicted LP solution. Concretely, we consider the acyclic reorientation $G_r^{\mathcal{A}} = (V_r, E_r^{\mathcal{A}}, r_r)$ with $E_r^{\mathcal{A}} = \{(i,k),(i,j),(j,k)\}$, such that i is the root, $r_r = i$. Assuming that i is initially mapped on node u_1, Algorithm 1 will map edges (i,k) and (i,j) first, setting $m_r^V(k) = u_6$ and $m_r^V(j) = u_2$ However, when the edge (j,k) is processed, k must be mapped on substrate node $u_3 \neq m_r^V(k)$ and the algorithm hence fails to produce a valid mapping. Accordingly, to avoid such *diverging* node mappings, our key idea is to decide the mapping location of nodes with more than one incoming edge (with respect to the request's acyclic reorientation) *a priori*.

By considering only cactus request graphs, this can be implemented rather easily as exactly one node of each cycle has more than one incoming edge: one only needs to ensure *compatibility* of node mappings for this node. To resolve potential conflicts for the mapping of this unique *cycle target*, our formulation employs *multiple* copies of the MCF formulation for the respective cycle subgraph. Specifically, considering a cycle with virtual *target* node k, we instantiate one MCF formulation per substrate node $w \in V_S^{r,k}$ onto which k can be mapped. Accordingly, this yields at most $|V_S|$ many copies and for each of these copies k is *fixed* to one specific (substrate) mapping location. Accordingly, as the mapping location of k is fixed to a specific node, valid mappings for the respective cycles can always be extracted from such a MCF copy: the mappings of k cannot possibly diverge.

A. Cactus Request Graph Decomposition and Notation

We decompose cactus request graphs as follows (cf. [17]).

Lemma 9. *Consider a cactus request graph G_r and its acyclic reorientation $G_r^{\mathcal{A}}$ of G_r. The graph $G_r^{\mathcal{A}}$ can be uniquely partitioned into subgraphs $\{G_r^{\mathcal{A},C_1}, \dots, G_r^{\mathcal{A},C_n}\} \sqcup G_r^{\mathcal{A},\mathcal{F}}$, s.t.:*

1) *The subgraphs $\{G_r^{\mathcal{A},C_1}, \dots, G_r^{\mathcal{A},C_n}\}$ correspond to the (undirected) cycles of G_r and $G_r^{\mathcal{A},\mathcal{F}}$ is the* forest *remaining after removing the cyclic subgraphs. We denote the index set of the cycles by $\mathcal{C}_r = \{C_1, \dots, C_n\}$.*
2) *The subgraphs partition the edges of $E_r^{\mathcal{A}}$: an edge $(i,j) \in E_r^{\mathcal{A}}$ is contained in exactly one of the subgraphs.*
3) *The edge set $E_r^{\mathcal{A},C_k}$ of each cycle $C_k \in \mathcal{C}_r$ can itself be partitioned into two branches $\mathcal{B}_1^{C_k}$ and $\mathcal{B}_2^{C_k}$, such that both lead from $s_r^{C_k} \in V_r^{\mathcal{A},C_k}$ to $t_r^{C_k} \in V_r^{\mathcal{A},C_k}$.*

Additionally, we denote by $G_r^{C_k}$ and $G_r^{\mathcal{F}}$ the subgraphs that agree with E_r on the edge orientations and use $V_{S,t}^{C_k} = V_S^{r,t_r^{C_k}}$ to denote the substrate nodes on which $t_r^{C_k}$ can be mapped.

B. Novel LP Formulation for Cactus Requests

Our novel Formulation 2 uses the a priori partition of $G_r^{\mathcal{A}}$ into cycles $G_r^{\mathcal{A},C_k}$ and the forest $G_r^{\mathcal{A},\mathcal{F}}$ to construct MCF formulations for the respective subgraphs: for the subgraph $G_r^{\mathcal{F}}$ a single copy is used (cf. Constraint 10) while for the cyclic subgraphs a single MCF formulation is employed *per* potential target location $V_{S,t}^{C_k}$ (cf. Constraint 11). We index the variables of these sub-LPs by employing square brackets.

To bind together these (at first) independent MCF formulations, we reuse the variables $\vec{x}, \vec{y}$, and $\vec{a}$ introduced already for the MCF formulation. We refer to these variables, which are defined outside of the sub-LP formulations, as *global variables* and do not index these. As we only consider the LP formulation, all variables are continuous.

The different sub-formulations are linked as follows. We employ Constraint 12 to enforce the setting of the (global) node mapping variables (cf. Constraint 2 of Formulation 1). By Constraints 13 and 14, the node mappings of the sub-LPs for mapping the subgraphs must agree with the global node mapping variables. With respect to cyclic subgraphs, we note that Constraint 14 allows for distributing the global node mappings to any of the $|V_{S,t}^{C_k}|$ formulations: only the sum of the node mapping variables must agree with the global node mapping variable. Constraint 15 is of crucial importance for the decomposability: considering the sub-LP for cycle C_k and target node $w \in V_{S,t}^{C_k}$, it enforces that the target node $t_r^{C_k}$ of the cycle C_k *must* be mapped on w. Thus, in the sub-LP $[C_k, w]$ both branches $\mathcal{B}_1^{C_k}$ and $\mathcal{B}_2^{C_k}$ of cycle C_k are *pre-determined* to lead to the node w. Lastly, for computing node allocations the global node mapping variables are used (cf. Constraint 16) and for computing edge allocations the sub-LP formulations' allocations are considered (cf. Constraint 17).

C. Decomposing Solutions to the Novel LP Formulation

We now show how to adapt the decomposition Algorithm 1 to decompose solutions to Formulation 2.

To decompose the LP solution for a request r the acyclic reorientation $G_r^{\mathcal{A}}$, which was also used for constructing the LP, must be handed over to the decomposition algorithm.

As the novel LP formulation does not contain (global) edge mapping variables, the edge mapping variables used in Lines 10 and 13 of Algorithm 1 must be substituted by edge mapping variables of the respective sub-LP formulations. Concretely, as each edge of the request graph G_r is covered exactly once, it is clear whether a virtual edge $(i,j) \in E_r$ is part of $G_r^{\mathcal{F}}$ or a cyclic subgraph $G_r^{C_k}$. If $(i,j) \in G_r^{\mathcal{F}}$ holds, then the edge mapping variables $z_{r,i,j}^{\cdot,\cdot}[\mathcal{F}_r]$ are used. If on the other hand the edge $(i,j) \in E_r$ is covered in the cyclic subgraph $G_r^{C_k}$, then there exist $|V_{S,t}^{C_k}|$ many sub-LPs to choose the respective edge mapping variables from. To ensure the decomposability, we proceed as follows.

If the edge $(i,j) \in E_r^{\mathcal{A}}$ is the first edge of $G_r^{C_k}$ to be mapped in the k-th iteration, the mapping variables $z_{r,i,j}^{\cdot,\cdot}[C_k, w]$ be-

Formulation 2: Novel LP for Cactus Requests

$$\max \sum_{r \in \mathcal{R}} b_r x_r \tag{9}$$

Cons. (2) - (7) for $G_r^{\mathcal{F}}$ on variables $(x_r, \vec{y}_r, \vec{z}_r, \vec{a}_r)[\mathcal{F}_r]$ $\quad \forall r \in \mathcal{R}$ (10)

Cons. (2) - (7) for $G_r^{C_k}$ on variables $(x_r, \vec{y}_r, \vec{z}_r, \vec{a}_r)[C_k, w]$ $\quad \forall r \in \mathcal{R}, C_k \in \mathcal{C}_r, w \in V_{S,t}^{C_k}$ (11)

$$x_r = \sum_{u \in V_S^{r,i}} y_{r,i}^u \qquad \forall r \in \mathcal{R}, i \in V_r \tag{12}$$

$$y_{r,i}^u = y_{r,i}^u[\mathcal{F}] \qquad \forall r \in \mathcal{R}, i \in V_r^{\mathcal{F}}, u \in V_S^{r,i} \tag{13}$$

$$y_{r,i}^u = \sum_{w \in t_r^{C_k}} y_{r,i}^u[C_k, w] \qquad \forall \begin{bmatrix} r \in \mathcal{R}, i \in V_r, u \in V_S^{r,i}, \\ C_k \in \mathcal{C}_r : i \in V_r^{C_k} \end{bmatrix} \tag{14}$$

$$0 = y_{r,t_r^{C_k}}^u[C_k, w] \qquad \forall \begin{bmatrix} r \in \mathcal{R}, C_k \in \mathcal{C}_r, w \in V_{S,t}^{C_k}, \\ u \in V_{S,t}^{C_k} \setminus \{w\} \end{bmatrix} \tag{15}$$

$$a_r^{\tau,u} = \sum_{i \in V_r, \tau_r(i) = \tau} d_r(i) \cdot y_{r,i}^u \qquad \forall r \in \mathcal{R}, (\tau, u) \in R_S^V \tag{16}$$

$$a_r^{u,v} = a_r^{u,v}[\mathcal{F}] + \sum_{C_k \in \mathcal{C}_r, w \in V_{S,t}^{C_k}} a_r^{u,v}[C_k, w] \qquad \forall r \in \mathcal{R}, (u,v) \in E_S \tag{17}$$

$$\sum_{r \in \mathcal{R}} a_r^{x,y} \leq d_S(x,y) \qquad \forall (x,y) \in R_S \tag{18}$$

longing to an arbitrary target node w, with $y_{r,i}^{m_r^V(i)}[C_k, w] > 0$, are used. Such a node w exists by Constraint 14.

If another edge (i', j') of the same cycle was already mapped in the k-th iteration, the *same* sub-LP as chosen before is considered. Accordingly, the mapping of cycle target nodes cannot conflict and as these are the only nodes with potential mapping conflicts, the returned mappings are always valid.

To successfully iterate the extraction process, the steps taken in Lines 18 - 21 of Algorithm 1 must be adapted to consider the sub-LP variables. Again, as in each iteration at least a single variable of the LP is set to 0 and as the novel Formulation 2 contains at most $\mathcal{O}(|V_S|)$ times more variables than the MCF Formulation 1, the decomposition algorithm still runs in polynomial-time. Hence, we conclude that the result of Lemma 6 carries over to the novel LP Formulation 2 for *cactus* request graphs and state the following theorem.

Theorem 10. *Given a solution $(x_r, \vec{y}_r, \vec{z}_r, \vec{a}_r)$ to the novel LP Formulation 2 for a cactus request graph G_r, the solution can be decomposed into a convex combination of valid mappings $\mathcal{D}_r = \{(m_r^k, f_r^k)\}_k$ in polynomial-time, such that:*

- *The decomposition is complete, i.e. $x_r = \sum_k f_r^k$ holds.*
- *The decomposition's resource allocations are bounded by $\vec{a}_r$: $a_r^{x,y} \geq \sum_k f_r^k \cdot A(m_r^k, x, y)$ holds for $(x, y) \in R_S$.*

IV. Approximation via Randomized Rounding

Above we have shown how optimal convex combinations for the VNEP can be computed for cactus requests. Given these convex combinations, the pseudo-code of our approximation for the VNEP is presented as Algorithm 2.

The algorithm first performs a preprocessing in Lines 1-3 by removing all requests which cannot be fully (fractionally) embedded in the absence of other requests, as these can never be part of any feasible solution. In Lines 4-6 an optimal solution to the novel LP Formulation 2 is computed and afterwards decomposed into convex combinations. Then, in Lines 7-9, the rounding is performed: for each request r a mapping m_r^k is selected with probability f_r^k. Importantly, the sum of probability may not sum to 1, i.e. with probability $1 - \sum_k f_r^k$ the request r is not embedded.

The rounding procedure is iterated as long as the constructed solution is not of sufficient quality or until the number of maximal rounding tries is exceeded. Concretely, we seek (α, β, γ)-approximate solutions which achieve at least a factor of $\alpha \leq 1$ times the optimal (LP) profit and exceed node and edge capacities by at most factors of $\beta \geq 1$ and $\gamma \geq 1$, respectively. In the following we derive parameters α, β, and γ for which solutions can be found *with high probability*.

Note that Algorithm 2 is indeed a polynomial-time algorithm, as the size of the novel LP Formulation 2 is polynomially bounded and can hence be solved in polynomial-time.

Algorithm 2: Randomized Rounding for the VNEP

```
1 foreach r ∈ R do                          // preprocess requests
2     compute LP Formulation 2 for request r maximizing x_r
3     if x_r < 1 then remove request r from the set R
4 compute LP Formulation 2 for R maximizing Σ_{r∈R} b_r · x_r
5 foreach r ∈ R do                          // perform decomposition
6     compute D_r = {(f_r^k, m_r^k)}_k from LP solution
7 do                                        // perform randomized rounding
8     foreach r ∈ R select m_r^k with probability f_r^k
9 while ( solution is not (α,β,γ)-approximate and
          maximal rounding tries are not exceeded )
```

A. Probabilistic Guarantee for the Profit

For bounding the profit achieved by the randomized rounding scheme, we recast the profit achieved in terms of random variables. The *discrete* random variable $Y_r \in \{0, b_r\}$ models the profit achieved by the rounding of request $r \in \mathcal{R}$. According to our rounding scheme, we have $\mathbb{P}(Y_r = b_r) = \sum_k f_r^k$ and $\mathbb{P}(Y_r = 0) = 1 - \sum_k f_r^k$. We denote the overall profit by $B = \sum_{r \in \mathcal{R}} Y_r$ with $\mathbb{E}(B) = \sum_{r \in \mathcal{R}} b_r \cdot \sum_k f_r^k$. Denoting the profit of an optimal LP solution by $\mathrm{B_{LP}}$, we have $\mathrm{B_{LP}} = \mathbb{E}(B)$ due to the decomposition's completeness (cf. Theorem 10).

By preprocessing the requests and confirming that each request can be fully embedded, the LP will attain at least the maximal profit of any of the considered requests:

Lemma 11. $\mathbb{E}(B) = \mathrm{B_{LP}} \geq \max_{r \in \mathcal{R}} b_r$ *holds.*

We employ the following Chernoff bound over continuous variables to bound the probability of achieving a small profit.

Theorem 12 (Chernoff Bound [19])**.** *Let $X = \sum_{i=1}^n X_i$, $X_i \in [0, 1]$, be a sum of n independent random variables. For any $0 < \varepsilon < 1$, the following holds:*

$$\mathbb{P}\big(X \leq (1 - \varepsilon) \cdot \mathbb{E}(X)\big) \leq \exp(-\varepsilon^2 \cdot \mathbb{E}(X)/2)$$

Theorem 13. *Let $\mathrm{B_{IP}}$ denote the profit of an optimal solution. Then $\mathbb{P}(B < 1/3 \cdot \mathrm{B_{IP}}) \leq \exp(-2/9) \approx 0.8007$ holds.*

Proof. Let $\hat{b} = \max_{r \in \mathcal{R}} b_r$ be the maximum benefit among the pre-processed requests. We consider random variables $Y_r' = Y_r / \hat{b}$, such that $Y_r' \in [0, 1]$ holds. Let $B' = \sum_{r \in \mathcal{R}} Y_r' = B/\hat{b}$.

As $\mathbb{E}(B) = \mathrm{B_{LP}} \geq \hat{b}$ holds (cf. Lemma 11), we have $\mathbb{E}(B') \geq 1$. Choosing $\varepsilon = 2/3$ and applying Theorem 12 on B' we obtain $\mathbb{P}\big(B' \leq (1/3) \cdot \mathbb{E}(B')\big) \leq \exp(-2 \cdot \mathbb{E}(B')/9)$. Plugging in the *minimal* value of $\mathbb{E}(B')$, i.e. 1, into the equation we obtain: $\mathbb{P}\big(B' \leq (1/3) \cdot \mathbb{E}(B')\ \big) \leq \exp(-2/9)$ and by linearity $\mathbb{P}\big(B \leq (1/3) \cdot \mathbb{E}(B)\ \big) \leq \exp(-2/9)$.

Denoting the profit of an optimal solution by $\mathrm{B_{IP}}$ and observing that $\mathrm{B_{IP}} \leq \mathrm{B_{LP}}$ holds as the linear relaxation yields an upper bound, we have $\mathrm{B_{IP}}/3 \leq \mathbb{E}(B)/3$. Accordingly, we conclude that, $\mathbb{P}\big(B \leq (1/3) \cdot \mathrm{B_{IP}}\big) \leq \exp(-2/9)$ holds. □

B. Probabilistic Guarantee for Resource Augmentations

In the following, we analyze the probability that a rounded solution exceeds substrate capacities by a certain factor.

We first note that $d_{\max}(r, x, y) \leq d_S(x, y)$ holds for all resources $(x, y) \in R_S$ and all requests $r \in \mathcal{R}$. We model the allocations on resource $(x, y) \in R_S$ by request $r \in \mathcal{R}$ as random variable $A_{r,x,y} \in [0, A_{\max}(r, x, y)]$. By definition, we have $\mathbb{P}(A_{r,x,y} = A(m_r^k, x, y)) = f_r^k$ and $\mathbb{P}(A_{r,x,y} = 0) = 1 - \sum_k f_r^k$. Furthermore, we denote by $A_{x,y} = \sum_{r \in \mathcal{R}} A_{r,x,y}$ the random variable capturing the overall allocations on resource $(x, y) \in R_S$.

$\mathbb{E}(A_{x,y}) = \sum_{r\in\mathcal{R}} \sum_k f_r^k \cdot A(r,x,y)$ holds by Theorem 10, we obtain $\mathbb{E}(A_{x,y}) \leq d_S(x,y)$ for all resources $(x,y) \in R_S$.

We employ Hoeffding's inequality to upper bound $A_{x,y}$.

Theorem 14 (Hoeffding's inequality [19])**.** *Let $X = \sum_{i=1}^{n} X_i$, $X_i \in [a_i, b_i]$, be a sum of n independent random variables. The following holds for any $t \geq 0$:*

$$\mathbb{P}(X - \mathbb{E}(X) \geq t) \leq \exp(-2t^2/(\sum\nolimits_i (b_i - a_i)^2))$$

Lemma 15. *Consider a resource $(x,y) \in R_S$ and $0 < \varepsilon \leq 1$, such that $d_{\max}(r,x,y)/d_S(x,y) \leq \varepsilon$ holds for $r \in \mathcal{R}$. Let $\Delta(x,y) = \sum_{r\in\mathcal{R}: d_{\max}(r,x,y)>0} (A_{\max}(r,x,y)/d_{\max}(r,x,y))^2$.*

$$\mathbb{P}(A_{x,y} \geq \delta(\lambda) \cdot d_S(x,y)) \leq \lambda^{-4} \tag{19}$$

holds for $\delta(\lambda) = 1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(x,y) \cdot \log(\lambda)}$ and any $\lambda > 0$.

Proof. We apply Hoeffding with $t = (1 - \delta(\lambda)) \cdot d_S(x,y)$:

$$\begin{aligned}
&\mathbb{P}(A_{x,y} - \mathbb{E}(A_{x,y}) \geq (1-\delta(\lambda)) \cdot d_S(x,y)) \\
&\leq \exp\Big(\frac{-4 \cdot \varepsilon^2 \cdot \log(\lambda) \cdot \Delta(x,y) \cdot d_S^2(x,y)}{\sum_{r\in\mathcal{R}} (A_{\max}(r,x,y))^2}\Big) \\
&\leq \exp\Big(\frac{-4 \cdot \varepsilon^2 \cdot \log(\lambda) \cdot \Delta(x,y) \cdot d_S^2(x,y)}{\sum_{r\in\mathcal{R}: d_{\max}(r,x,y)>0} (A_{\max}(r,x,y))^2}\Big) \\
&\leq \exp\Big(\frac{-4 \cdot \varepsilon^2 \cdot \log(\lambda) \cdot \Delta(x,y) \cdot d_S^2(x,y)}{\sum_{r\in\mathcal{R}: d_{\max}(r,x,y)>0} (\varepsilon \cdot d_S(x,y) \cdot A_{\max}(r,x,y)/d_{\max}(r,x,y))^2}\Big) \\
&\leq \exp\Big(\frac{-4 \cdot \log(\lambda) \cdot \Delta(x,y)}{\sum_{r\in\mathcal{R}: d_{\max}(r,x,y)>0} (A_{\max}(r,x,y)/d_{\max}(r,x,y))^2}\Big) = \lambda^{-4}
\end{aligned}$$

The second inequality holds, as $A_{\max}(r,x,y) > 0$ implies $d_{\max}(r,x,y) > 0$. For the third inequality, $A_{\max}(r,x,y) \leq \varepsilon \cdot d_S(x,y) \cdot A_{\max}(r,x,y)/d_{\max}(r,x,y)$ is used, which follows from the assumption $d_{\max}(r,x,y) \leq \varepsilon \cdot d_S(x,y)$ and $d_{\max}(r,x,y) > 0$. In the next step, $\varepsilon^2 \cdot d_S^2(x,y)$ is reduced from the fraction. As the denominator equals $\Delta(x,y)$ by definition, the final equality follows. Lastly, we utilize that the expected allocation $\mathbb{E}(A_{x,y})$ is upper bounded by the resource's capacity $d_S(x,y)$ to obtain Equation 19. □

Given Lemma 15, we obtain the following corollary.

Corollary 16. *Let $\varepsilon \leq 1$ be chosen minimally, such that $d_{\max}(r,x,y)/d_S(x,y) \leq \varepsilon$ holds for all resources $(x,y) \in R_S$ and all requests $r \in \mathcal{R}$. Let $\Delta(X) = \max_{(x,y)\in X} \Delta(x,y)$,*

$$\beta = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(R_S^V) \cdot \log(|V_S| \cdot |\mathcal{T}|)}) \text{ , and}$$
$$\gamma = (1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(E_S) \cdot \log(|E_S|)}) .$$

The following holds for all node resources $(\tau,u) \in R_S^V$ and edge resources $(u,v) \in E_S$, respectively:

$$\mathbb{P}(A_{\tau,u} \geq \beta \cdot d_S(\tau,u)) \leq (|V_S| \cdot |\mathcal{T}|)^{-4} \tag{20}$$
$$\mathbb{P}(A_{u,v} \geq \gamma \cdot d_S(u,v)) \leq |E_S|^{-4} \tag{21}$$

Proof. First, note that ε is chosen over all resources and requests and that $\Delta(R_S^V) \geq \Delta(\tau,u)$ and $\Delta(E_S) \geq \Delta(u,v)$ hold for $(\tau,u) \in R_S^V$ and $(u,v) \in E_S$, respectively. Equations 20 and 21 are then obtained from Lemma 15 by setting $\lambda = |V_S| \cdot |\mathcal{T}|$ for nodes and $\lambda = |E_S|$ for edges. □

C. Approximation Result

Given the probabilistic bounds established above, the main approximation result is obtained via a union bound.

Theorem 17. *Assume $|V_S| \geq 3$. Let β and γ be defined as in Corollary 16. Algorithm 2 returns (α,β,γ)-approximate solutions for the VNEP (restricted on cactus request graphs) of at least an $\alpha = 1/3$ fraction of the optimal profit, and allocations on nodes and edges within factors of β and γ of the original capacities, respectively, with high probability.*

Proof. We employ the following union bound argument. Employing Corollary 16 and as there are at most $|V_S| \cdot |\mathcal{T}|$ node resources and at most $|V_S|^2$ edges, the *joint* probability that any resource exceeds their respective capacity by factors of β or γ is upper bounded by $(|V_S| \cdot |\mathcal{T}|)^3 + |V_S|^2 \leq 1/27 + 1/9$ for $|V_S| \geq 3$. By Theorem 13 the probability of *not* finding a solution achieving an $\alpha = 1/3$ fraction of the optimal objective is upper bounded by $\exp(-2/9)$. Hence, the probability to not find a (α,β,γ)-approximate solution within a single round is upper bounded by $\exp(-2/9) + 1/9 + 1/27 \leq 19/20$. The probability to return a suitable solution within $N \in \mathbb{N}$ rounding tries is lower bounded by $1-(19/20)^N$ and Algorithm 2 yields approximate solutions for the VNEP *with high probability*. □

D. Discussion & Proposed Heuristics

Theorem 17 yields the first approximation algorithm for the profit variant of the VNEP. However, the direct application of Algorithm 2 to compute (α,β,γ)-approximate solutions is made difficult by the cumbersome definition of the terms $\Delta(R_S^V)$ and $\Delta(E_S)$. Specifically, computing β and γ exactly requires enumerating all valid mappings, which is not feasible. Hence, to directly apply Algorithm 2, the respective values have to be estimated. Considering $\Delta(R_S^V)$, the following upper bound can be easily established: $\Delta(R_S^V) \leq |\mathcal{R}| \cdot \max_{r\in\mathcal{R}} |V_r|$. However, plugging this bound into the definition of β yields rather large resource violations of $\beta \in \mathcal{O}(\varepsilon \cdot \sqrt{|\mathcal{R}| \cdot \max_{r\in\mathcal{R}} |V_r| \cdot \log(|V_S| \cdot |\mathcal{T}|)})$.

To overcome estimating β and γ, we propose the following:

Vanilla Rounding: A fixed number of solutions is rounded at random as in Line 7 of Algorithm 2. Afterwards, the best solution is returned according to some metric. In particular, in Section V we study the metric returning the solution of highest profit among the solutions minimizing the maximal resource augmentation.

Heuristical Rounding: In most settings resource augmentations are to be avoided based on their negative impact on the customer's Quality-of-Service. Hence, to obtain solutions not violating any resource's capacity, we propose to adapt the rounding scheme by simply *discarding* selected mappings, whose addition would exceed resource capacities. To increase the diversity of found solutions, the order in which requests are processed is permuted before each rounding iteration.

V. Explorative Computational Study

We now complement our formal approximation result in the standard multi-criteria model with resource augmentation

with an extensive computational study. Specifically, we study the performance of vanilla rounding and heuristical rounding without resource augmentations as introduced above.

As we are not aware of any systematic evaluation of the profit maximization in the offline settings, we present a synthetic but extensive computational study. Specifically, we have generated 1,500 offline VNEP instances with varying request numbers and varying demand-to-capacity ratios. For all instances, baseline solutions were computed by solving the Mixed-Integer Programming Formulation 1.

We restrict our discussion to our main results and refer the reader to our technical report at [17] for additional details. We have implemented all presented algorithms in Python 2.7 employing Gurobi 7.5.1 to solve Mixed-Integer Programs and Linear Programs. Our source code is freely available at [20]. All experiments were executed on a server equipped with Intel Xeon E5-4627v3 CPUs running at 2.6 GHz.

A. Instance Generation

We use the GÉANT topology[1] as substrate network. It consists of 40 nodes and 122 edges. We consider a single node type and set node and edge capacities uniformly to 100.

a) Request Topology Generation: Cactus graph requests are generated by (i) sampling a random binary tree of maximum depth 3, (ii) adding additional edges randomly as long as they do not refute the cactus property *as long as such edges exist*, and (iii) orienting edges arbitrarily.

We only consider requests containing at least 3 nodes. According to our generation parameters, the expected number of nodes and edges is 6.54 and 7.28, respectively. On average, 61% of the edges lie on a cycle.

b) Mapping Restrictions: To force the virtual networks to span across the whole substrate network, we restrict the mapping of virtual nodes to one quarter of the substrate nodes: each virtual node can be mapped on ten substrate nodes. The mapping of virtual edges is not restricted.

c) Demand Generation: We control the demand-to-capacity ratio of node and edge resource using a node resource factor NRF and an edge resource factor ERF. The request's demands are drawn from an exponential distribution and afterwards normalized, such that the following holds:

$$\sum_{r\in\mathcal{R}}\sum_{i\in V_r} d_r(i) = \mathrm{NRF}\cdot\sum_{u\in V_S} d_S(u)$$
$$\mathrm{ERF}\cdot\sum_{r\in\mathcal{R}}\sum_{(i,j)\in E_r} d_r(i,j) = \sum_{(u,v)\in E_S} d_S(u,v)$$

The resource factors can be best understood under the assumption that all requests are embedded. Under this assumption, a resource factor $\mathrm{NRF}=0.6$ implies that the node load – averaged over all substrate nodes – equals exactly 60%. As virtual edges can be mapped on arbitrarily long paths (even of length 0), the edge resource factor should be understood as follows: the ERF equals 'the number of substrate edges that each virtual edge may use'. In particular, a factor $\mathrm{ERF}=0.5$ implies that if *each* virtual edge spans *exactly* 0.5 substrate edges, then edge resource utilization equals exactly 100%. Hence, while increasing the NRF renders node resources more scarce, increasing the ERF reduces edge resource scarcity.

d) Profit Computation: To correlate the profit of a request with its size, its resource demands, and its mapping restrictions, we compute for each request its minimal embedding costs as follows. The cost $c(u,v)$ of using an edge $(u,v)\in V_S$ equals the geographical distance of its endpoints. The cost of nodes is set uniformly to $c(\cdot,u)=\sum_{(u,v)\in E_S} c(u,v)/|V_S|$ for all $u\in V_S$. Hence, the total node cost equals the total edge cost. Defining the cost of a mapping m_r to be $\sum_{(x,y)\in R_S} A(m_r,x,y)\cdot c(x,y)$, we compute the minimum cost embedding for each request $r\in\mathcal{R}$ using an adaption of Mixed-Integer Program 1 and set b_r accordingly.

e) Parameter Space: We consider the following parameters $|\mathcal{R}|\in\{40,60,80,100\}$, $\mathrm{NRF}\in\{0.2,0.4,0.6,0.8,1.0\}$, $\mathrm{ERF}\in\{0.25,\ 0.5,1.0,2.0,4.0\}$ and generate 15 instances per parameter combination, yielding $1,500$ instances overall.

B. Computational Results

We first present our baseline results and then study the performance of vanilla rounding and heuristical rounding.

a) Baseline $\mathrm{MIP}_{\mathrm{MCF}}$*:* To obtain a near-optimal baseline solution for each of the 1,500 instances, we employ Gurobi 7.5.1 to solve the Mixed-Integer Programming Formulation 1 (using a single thread). We terminate the computation after 3 hours or when the *objective gap* falls below 1%, i.e. when the constructed solution is *provably* less than 1% off the optimum. On average the runtime per instance is 129.8 minutes [17].

Figure 3 gives an overview on these baseline solutions. In particular, based on the a priori profit computation, the number of requests which can be feasibly embedded is shown together with the acceptance ratio which on average lies around 75%. The rightmost plot depicts the objective gap, i.e. the quality guarantee proven by Gurobi, which is (on average) 6.8%.

b) Solving LP Formulation 2: To apply the rounding algorithms presented in Section IV-D, our novel LP Formulation 2 needs to be solved. Again, we employ Gurobi 7.5.1, specifically its Barrier algorithm with crossover. Figure 4 depicts the averaged runtime to solve the LP as well as to construct the LP. The latter is not negligible as the formulation contains up to 1,000k variables for some instances. The runtime increases from around 2 minutes for $|\mathcal{R}|=40$ to around 7 minutes for $|\mathcal{R}|=100$. The maximally observed runtime in our experiments amounted to roughly 18 minutes.

c) Vanilla Rounding $\mathrm{RR}_{\mathrm{MinLoad}}$*:* We first consider the performance of vanilla rounding. Concretely, we report on the best solution found within 1,000 rounding iterations, i.e. the solution minimizing resource augmentations and breaking ties among these by returning the solution of highest profit. Figure 5 (left) depicts the results. As can be seen, the algorithm achieves a profit between 50% and 140% compared to the best solution constructed by the MIP, while exceeding resource capacities mostly by 25% to 125% of the resource's capacity. The edge resource factor has a distinctive impact: for $\mathrm{ERF}=4.0$ maximal resource loads mostly lie below 75%.

[1]Obtained from http://www.topology-zoo.org/ (version March 2012).

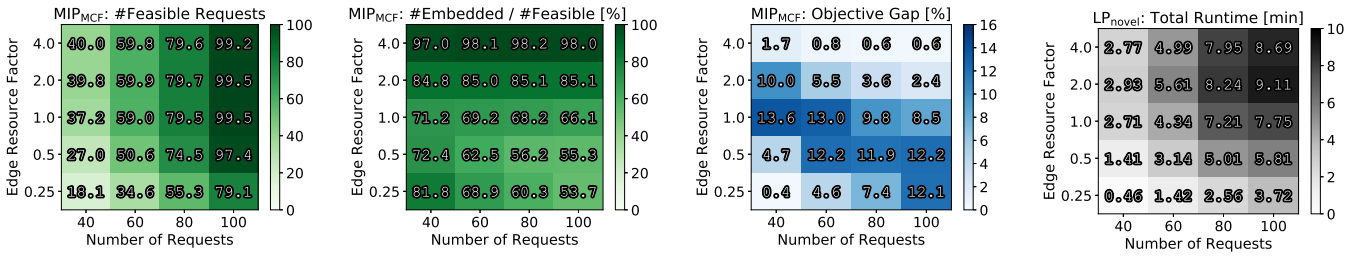


Fig. 3. Overview on baseline results computed using the MIP Formulation 1. Each cell averages the results over 75 instances. The feasibility of requests is obtained from (cost-optimally) embedding the requests to compute the profit a priori. The center plots depicts the acceptance ratio restricted to the feasible requests. The solution's quality is depicted on the right: the gap heavily depends on the edge resource factor but is on average less than 7%.

Fig. 4. Solution time of the novel LP Formulation 2 (including the construction time for the LP) using the Barrier algorithm of Gurobi 7.5.1.

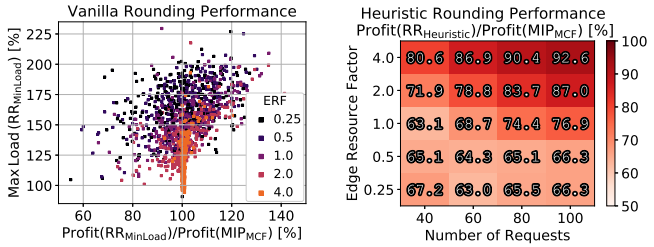


Fig. 5. Overview on results obtained using vanilla and heuristical rounding.
Left: Solutions obtained via *vanilla* rounding minimizing the load. Each point corresponds to a single instance and is colored according to the instance's edge resource factor. 7 of 1,500 results lie outside the depicted area.
Right: The averaged profit of solutions obtained via heuristical rounding compared to the best baseline solution. Each cell averages 75 instances.

d) Heuristical Rounding $\mathrm{RR}_{\mathrm{Heuristic}}$*:* The results of the heuristical rounding are presented in Figure 5 (right). Again, 1,000 rounding iterations were considered. While for low edge resource factors, i.e. scarce edge resources, the solutions achieve around 65% of the profit of the MIP baseline, for larger edge resource factors, the relative performance exceeds 80%. Furthermore, the performance improves when increasing the number of requests. Overall, the average relative performance with respect to the baseline solutions is 73.8%, with the minimal one being 22.3%.

VI. Conclusion

This paper has initiated the study of approximation algorithms for the Virtual Network Embedding Problem supporting arbitrary substrate graphs and supporting virtual networks containing cycles. To obtain the approximation, we have derived a strong LP formulation for cactus request graphs. Our computational evaluation shows the practical significance of our work: obtained solutions achieve (on average) around 74% of the baseline's profit while *not augmenting capacities*.

We note that the developed approximation framework is independent of the how LP solutions are computed and decomposed. In particular, while the LP formulation presented in this paper is only applicable for cactus request graphs, our formulation can be generalized to arbitrary request graphs [18].

Acknowledgements

This work was partially supported by Aalborg University's PreLytics project as well as by the German BMBF Software Campus grant 01IS1205.

We thank Elias Döhne, Alexander Elvers, and Tom Koch for their significant contribution to our implementation [20].

References

[1] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM CCR*, vol. 42, no. 5, 2012.
[2] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. 3rd IEEE CloudNet*, October 2014.
[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.
[4] J. Napper, W. Haeffner, M. Stiemerling, D. R. Lopez, and J. Uttaro, "Service Function Chaining Use Cases in Mobile Networks," Internet-Draft, Apr. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06
[5] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Comm. Surveys Tutorials, IEEE*, vol. 15, no. 4, 2013.
[6] M. Rost and S. Schmid, "Charting the Complexity Landscape of Virtual Network Embeddings," in *Proceedings IFIP Networking*, 2018.
[7] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE INFOCOM*, 2009.
[8] R. Hartert et al., "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *SIGCOMM*, 2015.
[9] A. Jarray and A. Karmouch, "Decomposition approaches for virtual network embedding with one-shot node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 1012–1025, 2015.
[10] M. Rost, S. Schmid, and A. Feldmann, "It's About Time: On Optimal Virtual Network Embeddings under Temporal Flexibilities," in *Proc. IEEE IPDPS*, 2014, pp. 17–26.
[11] M. Rost, C. Fuerst, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, 2015.
[12] G. Even, M. Medina, and B. Patt-Shamir, "Online path computation and function placement in sdns," in *Proc. International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2016.
[13] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in sdns," in *Proc. SIROCCO*, 2016.
[14] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proc. 22nd SIROCCO*, 2015.
[15] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, "Minimum congestion mapping in a cloud," in *Proc. ACM PODC*, 2011.
[16] M. Rost and S. Schmid, "Service chain and virtual network embeddings: Approximations using randomized rounding," Tech. Rep. arXiv:1604.02180 [cs.NI], April 2016.
[17] ——, "Virtual Network Embedding Approximations: Leveraging Randomized Rounding," Tech. Rep. arXiv:1803.03622 [cs.NI], March 2018.
[18] ——, "(FPT-)Approximation Algorithms for the Virtual Network Embedding Problem," Tech. Rep. arXiv:1803.04452 [cs.NI], March 2018.
[19] D. P. Dubhashi and A. Panconesi, *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
[20] E. Döhne, A. Elvers, T. Koch, and M. Rost, "Source code for the evaluation presented in this work," https://github.com/vnep-approx/evaluation-ifip-networking-2018.

AHAB: Data-Driven Virtual Cluster Hunting

Johannes Zerwas*, Patrick Kalmbach*, Carlo Fuerst†, Arne Ludwig†, Andreas Blenk*,
Wolfgang Kellerer*, Stefan Schmid‡
*Technical University of Munich, Germany †Technical University of Berlin, Germany ‡University of Vienna, Austria

***Abstract*—Virtual clusters are an important concept to provide isolation and predictable performance for multi-tenant applications in shared data centers. The problem of how to embed virtual clusters in a resource efficient manner has received much attention over the last years. However, existing virtual cluster embedding algorithms typically optimize the embedding of a *single* request. We demonstrate that this can lead to fragmentation and suboptimal data center resource utilization over time. We propose an alternative in two stages: First, we describe a novel embedding algorithm, called TETRIS, which, in an effort to avoid resource fragmentation over time, takes into account the specific node-to-link resource ratios of the individual requests. While TETRIS can be suboptimal when embedding only one request, we find that it performs much better than the state-of-the-art algorithms over time. Second, we allow the algorithm to strategically *reject* individual requests, even if there are sufficient resources: our proposed algorithm, AHAB, hence selects ("hunts") useful requests over time. An important property of AHAB is that it is *data-driven*: it uses information about previous requests and embeddings. We report on extensive simulations, which demonstrate the optimization potential of TETRIS (+4%) and AHAB (+13%), compared to existing solutions such as KRAKEN and OKTOPUS. Furthermore, AHAB illustrates how data-driven algorithms can replace man-made heuristics.**

***Index Terms*—Network Virtualization, Embedding, Admission Control**

I. INTRODUCTION

Today's data analysis frameworks and cloud applications generate large amounts of traffic; hence, their overall performance depends on the network. Indeed, it has been shown that cloud applications suffer from resource interference on the network, to the extent that the application execution times may become unpredictable [1]. To overcome this, several systems have been introduced that provide isolation among different customers and ensure network conditions as required by data center applications [2]–[6].

A common resource reservation abstraction provided to the tenant is the virtual cluster (VC) [2]. A VC connects a number of virtual machines (VM) to a virtual switch at a guaranteed bandwidth. The problem of how to embed virtual clusters has already received much attention [2], [7]–[10]. Proposed systems typically optimize the embedding of a single request: Minimizing the physical resource footprint of a single VC is often stated as the goal of the algorithms [7]–[9]. However, VC embedding is usually applied in an online environment where requests arrive over time. Focusing only on a single VC while neglecting the impact on future embeddings may fragment the reserved physical resources. This can in turn harm the resource utilization over time. Instead of looking only at single VCs, we propose to leverage information about the embedded request characteristics. Indeed, recent analysis of data center traces show that request characteristics can be estimated with sufficient quality to make scheduling decisions [11]; an invaluable source to optimize data center resource utilization. By integrating information about VCs into the embedding decision, this work makes two steps to surpass the drawbacks of existing VC embedding algorithms.

First, we present a novel embedding algorithm, TETRIS, which aims to reduce fragmentation over time by accounting for the ratio of requested node and link resources (which can differ from request to request), compared to the available resources in the substrate.

Second, we extend our study to admission control algorithms: we allow algorithms to strategically reject requests even though the substrate would provide enough resources to host the current to-be-embedded VC. In particular, this paper proposes AHAB[1] — a data-driven approach to admission control. The key idea of the data-driven paradigm is to base the decisions on observations from collected data instead of relying on manually designed strategies and it has recently drawn attention in networking research [12].

AHAB exploits knowledge about the characteristics of VCs. More specifically, it uses distributions of the VC attributes (VMs, bandwidth) to generate requests and evaluate the impact of the new VC on the feasibility of future embeddings. Concretely, AHAB answers the question whether the current VC will negatively affect the data center utilization in the future. To do so, AHAB performs several small simulations and compares their outcomes for two cases: one where the new VC is accepted and one where it is not. As it relies on a data-driven concept only, AHAB is independent from embedding algorithms, i.e., any VC embedding algorithm can be used in combination with AHAB. Hence, it can easily extend existing cluster management systems.

Simulations show that TETRIS outperforms state-of-the-art single-request embedding algorithms, enabling providers to host more VCs and hence use their infrastructure more efficiently. Moreover, the evaluation demonstrates that data-driven admission control can greatly improve the resource utilization in data centers by integrating knowledge about the distributions of the requests' attributes into the admission decision. Even when facing mismatched distributions, AHAB provides higher cluster utilizations than existing algorithms.

[1]The name AHAB refers to Moby Dick's captain Ahab, hunting sea monsters like KRAKEN or OKTOPUS (the systems upon which AHAB improves).

ISBN 978-3-903176-08-9

II. Revisiting Virtual Cluster Embeddings

This section describes our considered scenario and VC abstraction. We also revisit the VC embedding problem, and list two state-of-the-art algorithms to optimize the embedding of a single VC.

A. Virtual Cluster Abstraction: State-of-the-art

Virtual clusters [2] are the most prominent abstraction for batch-processing applications. Using VC abstraction, tenants can specify their networking demands, which introduces predictable performance guarantees. A VC request consists of the number of VMs and the bandwidth that should be reserved for each VM. If the provider embeds the request, it creates the number of equally-sized VMs and allocates them on the hosts of the substrate network. Additionally, the provider creates bandwidth reservations on the physical links such that every VM can use the requested bandwidth. Hence, the tenant is provided with the illusion of a dedicated network.

Besides this basic abstraction, extended versions have been proposed [6], [7], [10], [13]. However, existing algorithms do not specifically account for the fact that different requests can have different ratios of node and link resources: virtual cluster specifications are likely to come with different requirements [14], e.g., some requests have high requirements for computational resources but do not transfer much data while others are more network-intensive and require less computational resources.

B. Scenario Description

Table I summarizes the mathematical names and conventions in notation that are used throughout this study.

Substrate. The considered substrate networks (physical cluster) $\mathcal{C}$ hosting VCs employ a tree-like topology, e.g. Fat-Tree [15], a common data center architecture today. A set of pods is connected via core switches. Each pod consists of several racks which are connected to the aggregation switch of the pod. The racks are constituted by several hosts (or servers) that are interconnected by the top of rack (ToR) switch. The capacities of the links of the aggregation levels equal the accumulated bandwidths of the corresponding child nodes. The computational size of a physical server is measured in integer-valued compute units (CU). Similarly, the capacity of the physical hosts' links are normalized to denote the bandwidth in integer-valued bandwidth units (BU).

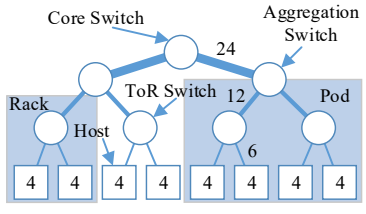
According to the physical cluster modeling in [2], [9], we approximate the Fat-Tree by a simple tree. The Fat-Tree depicted in Fig. 1a consists of two pods, containing two racks each; there are two hosts per rack. A host has a capacity of 4 CUs and the hosts' link capacities are 6 BUs. The links on aggregation and core level have capacities of 12 BUs and 24 BUs respectively.

Virtual Cluster. The VC abstraction should reflect the described observations from Sec. II-A. Customers should be able to specify their computation and communication requirements separately. Concretely, a VC is the triple $\mathcal{R} = (N, S, B)$, where N is the number of VMs, S is the computational requirement (size) of a VM and B is the bandwidth of a virtual link. All VMs are of the same computational size S, and are connected to a virtual switch at bandwidth B. For instance, the VC in Fig. 1b requests 3 VMs with a size of 4 CUs and a bandwidth of 2 BUs between the VMs and the virtual switch.

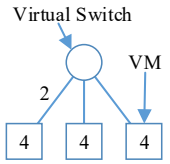
Online Cluster Arrival Process. Requests arrive in an online fashion and the provider must decide if a new request is embedded or rejected. In order to embed a VC, the provider has to fulfill all its specifications.

TABLE I
Notation and Abbreviations.

Symbol	Description
	Substrate
$\mathcal{C}$	Substrate network with a tree-like topology
CU	Compute unit: Abstract unit to measure computation requirements or capacity
BU	Bandwidth unit: Abstract unit to measure bandwidth requirements or capacity
C^h	Available computing capacity on host h [CU]
B^h	Available bandwidth on up-link of host h [BU]
FreeCapacity($\mathcal{C}$)	Number of free CUs in the substrate $\mathcal{C}$
TotalCapacity($\mathcal{C}$)	Total number of CUs in the substrate $\mathcal{C}$
Hosts($\mathcal{C}$)	Single hosts of $\mathcal{C}$ in groups of 1 sorted by avail. CUs.
Racks($\mathcal{C}$)	Hosts of $\mathcal{C}$ grouped by their racks
Pods($\mathcal{C}$)	Hosts of $\mathcal{C}$ grouped by their pods
Root($\mathcal{C}$)	Hosts of $\mathcal{C}$ in one large group
	Virtual Cluster Request
N	Number of VMs that a request has
S	Size of the VMs of a request [CU]
B	Bandwidth requirement per VM of a request [BU]
$\mathcal{R} = (N, S, B)$	Virtual cluster request with N VMs of size S interconnected with bandwidth B
VMs($\mathcal{R}$)	Virtual machines of request $\mathcal{R}$
host(vm)	Host which is assigned to the VM vm or $NULL$ if no host is assigned
$\rho(h, \mathcal{R})$	$= \frac{C^h - S}{B^h - B}$, ratio of available resources on host h after allocating one VM of request $\mathcal{R}$



(a) Fat-Tree with two pods, two racks per pod and two hosts per rack.



(b) VC with $N = 3$, $S = 4$ and $B = 2$.

Fig. 1. Examples for Fat-Tree and VC.

C. Existing VC Embedding Algorithms

In this study, we focus on two prominent VC embedding algorithms: Oktopus and Kraken.

Oktopus. Ballani et al. [2] proposed a first algorithm (henceforth called Oktopus) to embed VCs in Fat-Tree datacenter topologies. Its heuristic approach aims at minimizing

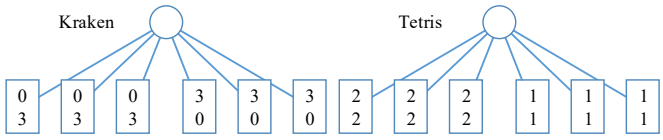


Fig. 2. Embedding behavior of KRAKEN and TETRIS. Six hosts (6 CUs, 6 BUs) are connected to a switch. Requested VCs are $\mathcal{R}^1 = (9, 1, 2)$, $\mathcal{R}^2 = (9, 2, 1)$. The upper number in a host represents mapped VMs of $\mathcal{R}^1$ and the lower number those of $\mathcal{R}^2$.

the allocation costs, but it does not always achieve optimal results. OKTOPUS iterates through the levels of the tree and searches for the first group of hosts (single host, hosts of a rack, hosts of a pod, all hosts) where the VC is feasible.

KRAKEN. An optimal solution to the single request embedding problem has been presented in [9], as part of the KRAKEN system. In contrast to OKTOPUS, KRAKEN returns the embedding with the minimal allocation cost for the given request and cluster state. To do so, KRAKEN does not return the first feasible solution, but checks all feasible solutions and returns the optimal solution for the VC. To maintain linear complexity w.r.t. the number of physical hosts, it uses the center of gravity concept, which corresponds to the location of the abstract virtual switch. It further allows to modify the size of the VC online.

Both algorithms (as well as algorithms lying between the two, like Proteus [7]) focus on single VCs and serve as comparables to the embedding algorithm presented in this study, which sacrifices quality of single embeddings to obtain better overall results. To the best of our knowledge, the challenge of fragmentation over time and the systematic study of the benefits of admission control has not been considered in the literature before.

III. TETRIS: ON THE POTENTIAL OF NON-GREEDY VC EMBEDDING

TETRIS is a VC embedding algorithm that, in an effort to perform better over time and in the long run, accounts for the specific resource ratios (and hence potential undesired resource fragmentations). As we will see, despite its simplicity, TETRIS already outperforms state-of-the-art VC embedding algorithms, which do not account for such fragmentation over time.

A. Key Idea: Sacrificing Footprint for Fragmentation

The main idea is to utilize the different resource dimensions of single hosts in a more balanced fashion, in order to avoid fragmentation (and hence poor resource utilization) over time. OKTOPUS and KRAKEN find embeddings which are dense and use low amount of bandwidth. The problem of such dense embeddings is that requests with a resource ratio $S/B \neq 1$ are collocated which wastes physical resources. Fig. 2 gives an example. For KRAKEN, $\mathcal{R}^1$ is embedded on the right three hosts. Thus, there are 3 CUs left on the host but no capacity on the up-link, which renders it unlikely that those free CUs are used in the future. On the other hand, $\mathcal{R}^2$ only uses half of the link capacity of the left three hosts. Using TETRIS, the VMs of both VCs are distributed over the hosts more evenly such that no host has capacity left for only one resource.

The ratio $\rho(h, \mathcal{R}) = (C^h - S)/(B^h - B)$ serves as a score to determine the placement of the VMs, i.e., TETRIS prefers hosts with ratio. C^h and B^h are the currently available resources on host h. As C^h is in the nominator, hosts that have much compute resources available, but only little bandwidth, are more likely chosen as location for the next VM than hosts with few compute resources available and much bandwidth.

B. Algorithm Details

Algorithm 1 shows the procedure of TETRIS. After checking the general feasibility of the request (l.1f), the algorithm iterates over the levels of the tree topology starting at the host level (similar to OKTOPUS and KRAKEN).

Trying Hosts. TETRIS iterates over every single host and tries to place all requested VMs on the same host (ll.4-9). At this stage, the resource ratio does not matter since $hostGroup$ has only one element (l.5). If the request fits on a single host, no bandwidth reservation is needed and TETRIS returns.

Trying Racks/Pods/Cluster. If the request does not fit on a single host, TETRIS iterates over the racks of the cluster (sorted by the fraction of available compute capacity). But instead of collocating as many VMs on a single host as possible (like OKTOPUS and KRAKEN), TETRIS distributes the VMs over several hosts depending on the ratio of the residual resources per host. For each VM of the request, TETRIS chooses the feasible host with the highest ratio $\rho(h, \mathcal{R})$ as location from all hosts of the current rack (l.5). Then TETRIS reserves resources for the VM on the host and the hosts' link (l.9). Bandwidth reservations on higher layers (aggregation, core) cannot be performed because the location of the virtual switch is not known yet. If any of the VM allocations fails, TETRIS resets the previously allocated VMs and proceeds with the hosts of the next rack (ll.6-8) — no host of the current rack will be used.

When all VMs are placed, TETRIS determines the virtual switch's location and performs the final bandwidth reservations (l.10f). The previous steps (l.4-9) do not guarantee the feasibility of the bandwidth reservations on the aggregation and core layer and the reservations may fail, e.g., if the Fat-Tree is oversubscribed. If this is the case, TETRIS removes the embeddings of the VMs and starts over using the hosts of the next rack (l.14f).

The same procedure is applied for the pod and root levels, if the algorithm has not found any feasible embedding after having evaluated all racks. If TETRIS does not find any feasible solution, the VC is rejected. TETRIS' complexity is linear in the number of topology host like OKTOPUS and KRAKEN.

IV. AHAB: THE CASE FOR DATA-DRIVEN ADMISSION CONTROL

We now take the idea of thinking strategically and being less greedy in how a single request is embedded one step further and initiate the study of algorithms which can even

Algorithm 1 Virtual Cluster Embedding: TETRIS

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$
Output: Embedding success

if FreeCapacity($\mathcal{C}$) $< N \cdot S$ **then**
  return False
for $hostGroup \in$ {Hosts($\mathcal{C}$), Racks($\mathcal{C}$), Pods($\mathcal{C}$), Root($\mathcal{C}$)} **do**
  for $vm \in$ VMs($\mathcal{R}$) **do**
    host(vm)$\leftarrow \arg\max \rho(h, \mathcal{R}),\ h \in hostGroup$: vm is feasible on h
    if host(vm)$== NULL$ **then**
      Reset host(vm) $\forall vm \in$ VMs($\mathcal{R}$)
      Continue with next $hostGroup$
    Reserve S, B on host(vm)
  $success \leftarrow$reserveBandwidth($\mathcal{R}$)
  if $success$ **then**
    return True
  else
    Reset $\mathcal{C}$ and host(vm) $\forall vm \in$ VMs($\mathcal{R}$)
    Continue with next $hostGroup$
return False

reject individual requests entirely, although there are sufficient resources available.

A. Key Idea: Admission Control and Leveraging Data

The admission control algorithm AHAB (Algorithm 2) shall "hunt" for the best VCs to embed. It can be configured with many single-request embedding algorithms, including TETRIS. Similar to DeepMind's AlphaGo [16] and other Monte Carlo Tree Searches [17], AHAB performs a lookahead search to make its decision. The idea is to get the impact of the embedding of a new VC on future arrivals. To do so, AHAB uses knowledge about the distributions of the VCs' attributes N, S, B to generate potential sequences of requests and tries to allocate these along with the actually arrived VC. The data collected with these small simulations is then the basis for the decision. The knowledge can be easily obtained from past requests and as a first step, we expect perfect knowledge about the distributions of N, S, B, which is an acceptable assumption as recent work has shown [11].

B. Algorithm Details

AHAB starts with checking the feasibility of the request. If the cluster is only lightly loaded, AHAB accepts the request (l.3f). This step reduces computational efforts as the probability of acceptance is high in this situation. If current load is $> 50\%$, AHAB performs the lookahead search. Given the current substrate state $\mathcal{C}$ and the new VC $\mathcal{R}$, AHAB generates a number of sequences ($numSeq$) of length $numVCs$ containing possible future requests and embeds them using the embedding algorithm A, e.g. KRAKEN (Algorithm 3). One half contains $\mathcal{R}$ while the other half does not (l.5f). Each allocated VC gives a reward of $N \cdot S$. AHAB uses the accumulated reward of the single sequences as a performance indicator and

Algorithm 2 Admission Control: AHAB

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$, Embedding algorithm A, $numSeq$, $numVCs$, Distributions for N, S, B
Output: Decision: Accept=True, Reject=False

1: **if** A cannot find a feasible solution **then**
2: **return** False
3: **if** FreeCapacity($\mathcal{C}$)> 0.5 TotalCapacity($\mathcal{C}$) **then**
4: **return** True
5: $avgAccept$ = RunSequences($\mathcal{C}$, $\mathcal{R}$, A, $numSeq$, $numVCs$, true)
6: $avgReject$ = RunSequences($\mathcal{C}$, $\mathcal{R}$, A, $numSeq$, $numVCs$, false)
7: **return** $avgReject < avgAccept$

Algorithm 3 RunSequences

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$, Embedding algorithm A, $numSeq$, $numVCs$, $embedVC$
Output: Average reward per sequence

1: $rewards = \{\}$
2: **for** $i = 1$ **to** $numSeq$ **do**
3: $\mathcal{C}' \leftarrow$ Copy $\mathcal{C}$
4: **if** $embedVC$ **then**
5: A.embed($\mathcal{C}'$, $\mathcal{R}$)
6: $rewards[i] = 0$
7: **for** $j = 1$ **to** $numVCs$ **do**
8: $\mathcal{R}' \leftarrow$ Generate new VC
9: **if** A.embed($\mathcal{C}'$, $\mathcal{R}'$) $==$ True **then**
10: $rewards[i] \mathrel{+}= S^{\mathcal{R}'} \cdot N^{\mathcal{R}'}$
11: **return** Average($rewards$)

determines the mean values for the sequences with and without $\mathcal{R}$ (Algorithm 3 - ll.9-11). The comparison of these two values gives the decision of acceptance (Algorithm 2 - l.7).

The complexity of AHAB depends on the complexity of the embedding algorithm and the total number of VC allocations ($= numVCs \cdot numSequences$) in one call, which are a tunable parameters.

V. EVALUATION

In order to analyze TETRIS and AHAB in different settings, we evaluate the results obtained using event-based simulations. Besides elaborating the differences in performance (Sec. V-B & V-C), we also look at the attributes of VCs that are accepted and try to understand why AHAB outperforms the other algorithms (Sec. V-D & V-E). Furthermore, we evaluate the impact of sequence length and number of sequences (Sec. V-F). This section closes with investigating the robustness of AHAB against errors in the distributions used to generate requests and varying host capacities (Sec. V-G & V-H).

A. Setup

Substrate. The physical cluster $\mathcal{C}$ is a three-layer Fat-Tree with construction number $k = 12$ resulting in 432 hosts in total. A host has a compute capacity of 8 CUs and 8 BUs on the connecting link which leads to a total of 3 456 CUs available in the cluster. The links between the ToR switches and the

aggregation switches and the links between the aggregation switches and the core are not oversubscribed.

Virtual Cluster Requests. The VCs arrive according to a Poisson process with an arrival rate λ and have exponentially distributed durations, such that they induce system load levels of 78.5% ($\lambda = 4$), 234% ($\lambda = 12$) and 390% ($\lambda = 20$). Based on the analyses of traces from Microsoft [11] and Google [18], the number of requested VMs N is exponentially distributed with mean 20 in the interval $[3, 60]$. B and S both follow a discrete distribution with $P(1) = 0.45, P(2) = 0.3, P(4) = 0.2, P(8) = 0.05$. All outcomes are sampled independently. We run every setup 30 with 1000 arriving VCs. To avoid artifacts related to the initially empty data center, we start evaluating our metrics after 100 requests.

Metrics. Various works [7], [10], [13] have used the acceptance ratio of an embedding algorithm in order to measure its performance. This metric, however, is biased towards algorithms that accept a large number of small requests instead of few bigger ones. Therefore, the first objective of this analysis is the maximization of the used CUs in the substrate. One sample is the average of this fraction over a whole run.

As second objective the minimization of the footprint $F(VC)$ of the embedded VCs is evaluated. The footprint of a VC is the amount of bandwidth of that VC that is reserved on the physical links (see Fuerst et al. [9]). For instance, the VC in Fig. 1 occupies one host per VM. The optimal embedding fills up one rack and uses one host of another rack. Bandwidth reservations are made on 5 physical links and $F(VC) = 10$.

Baseline Algorithms. The evaluation compares TETRIS with OKTOPUS and KRAKEN, all without admission control. Besides setups without admission control, the benchmark of AHAB also takes an observation-based strawman algorithm (STRAWMAN) into consideration, which simply rejects all VCs with $B > 4$ and uses KRAKEN to embed the VCs. This approach is based on the observations from TETRIS.

B. Is it worth playing TETRIS?

Fig. 3 compares the embedding algorithms for different arrival rates with admission control (AHAB with OKTOPUS, KRAKEN, TETRIS and STRAWMAN) and without admission control (OKTOPUS, TETRIS, KRAKEN). It shows the mean values over 30 runs with 95% confidence intervals. Unless otherwise stated, AHAB runs 20 sequences of 15 VCs.

Fig. 3a shows the cluster utilization w.r.t. CUs, the main objective of TETRIS and AHAB. For $\lambda = 4$, the cluster is not overloaded and all algorithms achieve similar values around 0.6. However, higher system loads, e.g., for $\lambda = 12$, allow to be more selective and make differences in the algorithms' performances visible. Considering TETRIS first, we observe that it outperforms OKTOPUS and KRAKEN for $\lambda \geq 12$ and achieves a mean CU usage of 0.83.

Yet, Fig. 3b suggests that this improvement does come at a certain cost. It shows the mean footprint of a single VC, i.e., the average number of physical link resources that are reserved for one VC. Generally, the mean values decrease with increasing arrival rates. For small arrival rates, the footprints obtained by OKTOPUS, KRAKEN and TETRIS are in a similar range (≈ 75 BUs), but for arrival rates around 12 the values for TETRIS are larger. This emphasizes the approach of TETRIS to sacrifice the footprint of the VCs to improve the utilization of the substrate. However, we observe that the gap between the average footprints of TETRIS and OKTOPUS and KRAKEN decreases further as the arrival rate increases towards 20.

The reason for this is highlighted by Fig. 3c, which shows the average number of concurrently embedded VCs. For all algorithms, this number significantly rises from 50 to ≈ 90 when the system transitions into overload and then only slightly increases further for OKTOPUS and KRAKEN. For TETRIS, it continues to grow with the arrival rate to values around 110 even though, the fraction of used CUs does not increase that much for arrival rates around 20. This implies that the average number of CUs per embedded VC decreases, i.e., TETRIS allocates more aggressively only small requests for high arrival rates while OKTOPUS and KRAKEN behave more moderately. A more detailed analysis follows later.

STRAWMAN pushes the performance of KRAKEN up and achieves cluster utilization values slightly worse than those of TETRIS (Fig. 3a). The VC footprints are smaller since requests with large bandwidth requirements are rejected; otherwise the minimal footprint for a VC is obtained. STRAWMAN trades off resource efficient embeddings with cluster utilization.

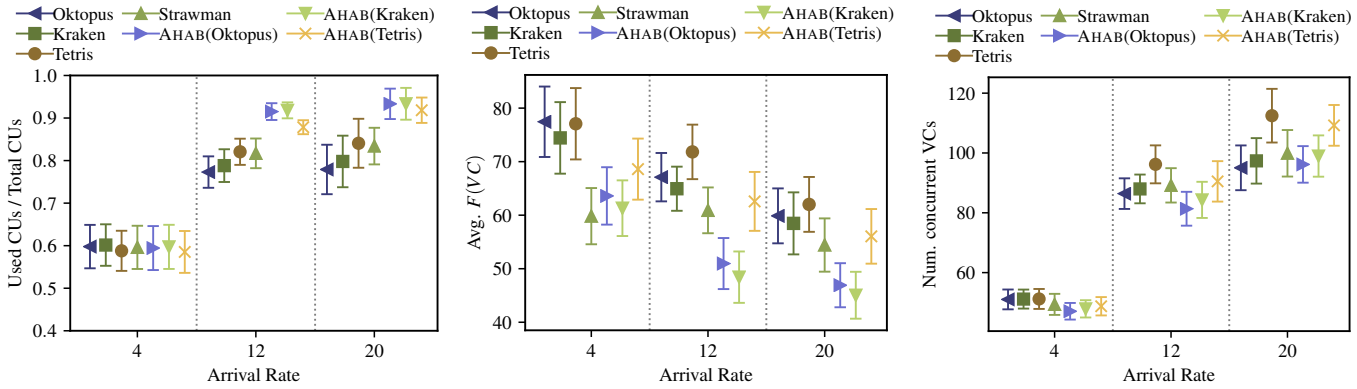
C. How useful is knowledge?

Adding admission control significantly improves the performance in terms of mean fractions of used CUs (Fig. 3a), when the system is overloaded ($\lambda \geq 12$). OKTOPUS and KRAKEN in combination with AHAB perform similar and the utilization exceeds 90%. A detailed explanation why both outreach TETRIS follows in Sec. V-F. Considering the average footprint of a single VC, AHAB(OKTOPUS) and AHAB(KRAKEN) show the best results with an average < 50 BUs for $\lambda \geq 12$. The other algorithms obtain mean values > 55 BUs. For $\lambda = 4$, the STRAWMAN dominates. AHAB(TETRIS) again results in increased footprints compared to AHAB(OKTOPUS) and AHAB(KRAKEN), e.g., for $\lambda = 12$, the mean value is ≈ 62 BUs. Fig. 3c suggests that AHAB also accepts more smaller requests as the number of concurrent VCs continues to rise with the arrival rate; however, it increases less than TETRIS without admission control.

To summarize, TETRIS improves the utilization of the substrate but is outperformed by solutions that incorporate admission control based on knowledge of the request generation process, i.e., the distributions of the VC attributes N, S, B. TETRIS is a credible alternative in case no knowledge is available or inaccessible.

D. Why is AHAB better?

In order to shed light on the reason behind AHAB's good performance, we look at the acceptance patterns of the algorithms. Fig. 4 visualizes the acceptance ratio of different request sizes for KRAKEN, TETRIS and AHAB(KRAKEN). The values are obtained from all requests of all runs.



(a) CU utilization (b) VC footprint (c) Num. concurrent VCs

Fig. 3. Performance comparison between embedding algorithms without admission control, with strawman admission control (only KRAKEN) and with AHABagainst the different requests' arrival rates. The subfigures show results for the fraction of used CUs, the average weighted footprint of a VC and the number of concurrently allocated VCs. The figures show the mean values with the 95% confidence intervals.

KRAKEN The color of a pixel corresponds to the acceptance ratio derived from the requests with that size. For instance, the upper left pixel of the block $B = 1$ in Fig. 4a means that KRAKEN accepts 80% of the requests with VM size $S = 1$, bandwidth requirement $B = 1$ and $3 \leq N < 9$. Somehow intuitive, higher acceptance ratios show up for smaller requests and the values decrease for larger requests. Especially for $B = 8$ or $S = 8$, KRAKEN is not able to embed many requests, as these occupy a whole host link or host. Still, KRAKEN allocates some of these requests. A higher number of requested VMs also decreases the acceptance ratio. For small VM sizes ($S \leq 2$), this effect is moderated by the requested bandwidth: For $B \leq 2$, the acceptance ratio is ≥ 0.4 for all bins of Num. VMs. For $B = 4$, the acceptance drops for requests with more than 33 VMs and for $B = 8$, the acceptance already drops to 0.2 for requests with 10 VMs.

TETRIS Fig. 4b shows the same representation for TETRIS. It supports the observations from Sec. V-B. Generally, TETRIS obtains higher acceptances ratios for VCs with small VM sizes and lower number of VMs. For instance, the acceptance ratio is ≥ 0.6 for requests with $B = 1$ and $S = 1$ regardless of N, while KRAKEN achieves these ratios only for requests with less than 27 VMs. But KRAKEN allocates more requests that occupy whole hosts or host links ($S = 8$ and $B = 8$). In particular for $B = 8$, the acceptance ratio of TETRIS and is less or equal to that of KRAKEN for almost all cases. This observation is the basis of the strawman admission control algorithm that was introduced before. Additionally for $S = 8$, TETRIS allocates only $\leq 40\%$ of the requests.

AHAB The behavior of TETRIS might not be optimal, as TETRIS does not perform best among the algorithms. Indeed, AHAB(KRAKEN) selects different VCs as Fig. 4c illustrates. For small bandwidths ($B = 1$), AHAB admits and embeds at least as many requests as KRAKEN without admission control. For $B > 1$, we observe that it embeds less requests with small VMs ($S = 1$). In this case, the acceptance ratio drops below 20% for requests with more than 15 VMs. But for VCs with larger VMs, AHAB obtains an acceptance ratio that is $5 - 10$ percentage points higher compared to KRAKEN in many cases.

In conclusion, TETRIS and AHAB increase the utilization of the substrate network but employ different acceptance patterns to do so.

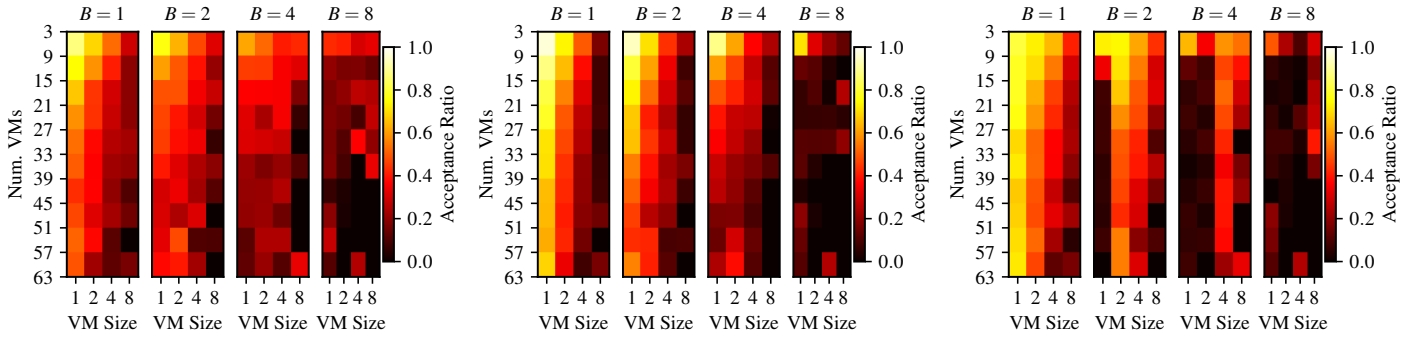
E. Which requests are valuable?

To understand why AHAB's acceptance pattern performs better than that of TETRIS, we look at the acceptance ratio from a different point of view: the value of a VC to the cluster utilization. Fig. 5 shows the acceptance ratio grouped by the resource ratio $\tilde{\rho} = \frac{S}{B}$ of a request and compares the values for KRAKEN and TETRIS without admission control and AHAB(KRAKEN). The previous observation is only weakly affected by the number of VMs in a request, which allows to reduce the dimensionality of the representation.

Small ratios mean that the allocation increases the target metric (used CUs) only little while occupying many network resources. Regardless the difference in absolute values, we note that KRAKEN and TETRIS have higher acceptance for VCs with $\tilde{\rho} \leq 1$ and reject VCs with $\tilde{\rho} > 1$ more likely. This is somehow counterintuitive as the benefit is low, while the probability for high allocation costs is high and further reflects that no explicit admission control is performed. In contrast to this, AHAB(KRAKEN) picks VCs with $\tilde{\rho} > 1$ as indicated by the acceptance ratios. For $\tilde{\rho} < 1$, the average acceptance ratio is 0.23, while it is 0.45 for VCs with $\tilde{\rho} > 1$. Thus, AHAB(KRAKEN) admits more valuable requests and thereby compensates the drawbacks of OKTOPUS and KRAKEN in comparison to TETRIS.

F. Optimization Opportunities: Can we save data?

The parameters that AHAB has used up to now ($numVCs = 15$, $numSeq = 20$), obtain the best results. However, it is generally desirable to minimize the computational overhead of AHAB. Therefore, we analyze the impact of the number of requests per sequence ($numVCs$) on AHAB's performance and also evaluate how sensitive the results are against the number of sequences ($numSeq$) that AHAB runs.



(a) KRAKEN (b) TETRIS (c) AHAB(KRAKEN)

Fig. 4. Comparison of acceptance ratio separated by VC specification. Each pixel of the heatmaps shows the value for the corresponding group of VCs. Subfigures allow to compare KRAKEN, TETRIS and AHAB(KRAKEN) and illustrate the differences in selection behavior. Arrival rate is 20. Note that Num. VMs is grouped into bins of size 6.

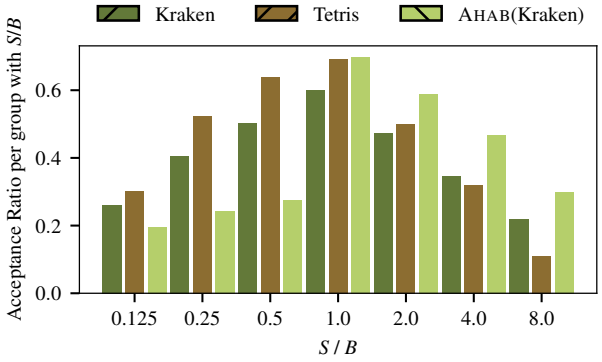
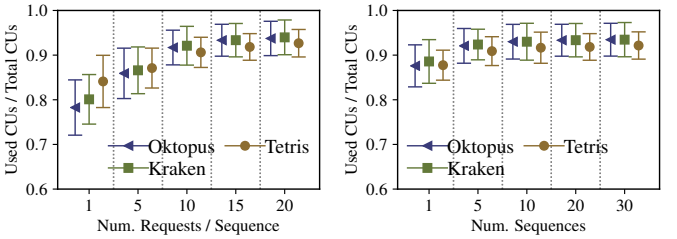


Fig. 5. Acceptance ratio per group of VCs with same resource ratio. Comparison between KRAKEN, TETRIS and AHAB(KRAKEN). Arrival rate is 20.



(a) Num. requests / sequence(b) Num. sequences ($numVCs = 15$) ($numSequences = 20$)

Fig. 6. Fraction of used CUs obtained by AHAB against the two parameters of AHAB. Comparison of results for OKTOPUS, KRAKEN and TETRIS. The sequence length positively affects the metric. 20 sequences containing 15 requests are enough for a high substrate utilizations ($> 90\%$).

Furthermore, we assess if the embedding algorithm affects the performance of AHAB.

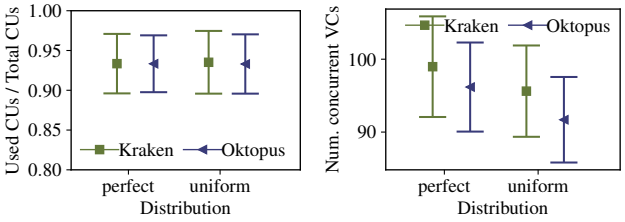
Fig. 6a visualizes how the fraction of used CUs changes with the number of requests per sequence. It shows the performance of AHAB admission control for all three embedding algorithms (OKTOPUS, KRAKEN and TETRIS). The number of sequences is fixed to 20. First, we observe that the utilization is positively affected by AHAB's sequence length. It grows from 0.8 for $numVCs = 1$ to 0.93 for $numVCs = 20$ and KRAKEN. The benefit of adding more requests vanishes as the sequences become longer. Additionally, the differences between the three embedding algorithms do not vary significantly for $numVCs > 5$. However, for small sequence lengths, the inherent performance of the embedding algorithm dominates: TETRIS is better than OKTOPUS and KRAKEN. The difference diminishes with increasing $numVCs$ and the break even is around 5 requests per sequence where AHAB produces the same utilization for all three embedding algorithms. For longer sequences the allocation with OKTOPUS or KRAKEN leads to higher substrate utilization. The implicit selection that TETRIS performs, limits the improvement, but the use of admission control still raises the fraction of used CUs by 0.08.

Fig. 6b shows in a similar way how the number of sequences affects the performance of AHAB. The sequence length is fixed to $numVCs = 15$. Except for the steps from 1 to 5 and from 5 to 10 sequences, we observe only very little change with increasing number of sequences. Thus, $10 - 20$ sequences are sufficient to obtain good results with AHAB. More sequences do not increase the quality of the decisions. This conclusion is not affected by the embedding algorithm.

In summary, looking more steps into the future improves the performance of AHAB at the cost of computation time. However, very long sequences do not further increase the substrate utilization, which allows to find good trade-offs. TETRIS performs an implicit selection of requests and its interference with AHAB leads to worse results compared to KRAKEN and OKTOPUS. The performance is not sensitive to the number of sequences which suffices to be in the range of several 10's.

G. What is the estimation error AHAB can cope with?

Sec. IV we assumed perfect knowledge about the generation process of VCs. This section relaxes the preceding assumption and evaluates how AHAB behaves, when it uses different distributions for generating the requests. The modified distributions have the same support as the original ones, but have a uniform shape.



(a) CU utilization (b) Num. concurrent VCs

Fig. 7. Comparison of AHAB's performance between different distributions for request generation through 95% confidence intervals. In both figures, the left group shows the results for the distributions as described in Sec. V-A. The right group uses uniform distributions with the same boundaries.

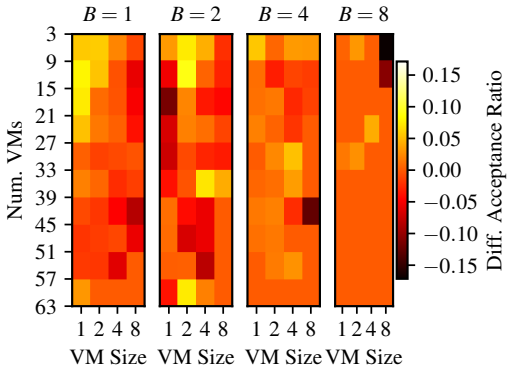


Fig. 8. Difference of acceptance ratio between AHAB(KRAKEN) with perfectly matched distributions and with mismatched/uniform distributions for request generation. Values are grouped by VC specification. Positive values show higher acceptance with matched distributions.

Fig. 7 compares how the substrate utilization and the number of concurrent VCs are affected by this change. The left group shows the results for perfect distribution estimation while the right ones contain the results obtained with the uniform distributions. AHAB runs 20 sequences with 15 requests each. Considering the fraction of used CUs, we observe that using a uniform distribution has no impact on AHAB.

However, Fig. 7b emphasizes that less VCs are embedded concurrently. This implies that larger requests are admitted by AHAB. An explanation for this is that using a uniform distribution instead of a geometric one results in a higher mean values of N, S, B. The mean number of VMs per request increases from 20 to 31.5 and the means for the bandwidth and VM size rise from 2.25 to 3.75. The higher mean value leads to an overestimation of the rewards obtained from future requests, when AHAB calculates the score for a sequence. As a consequence, it is less likely that the mean score of the sequences with accepted request is larger than the mean score of the sequences without the request. This is especially the case, when the arriving request is small and leads to more rejected small requests and a slightly higher number of accepted larger requests.

Fig. 8 underlines this. It shows the difference in acceptance ratio of the runs with perfect distribution estimation by AHAB and the runs with the uniform distribution used for request generation. In particular for $B = 1$, we observe that there are several groups of small requests with higher acceptance ratio when AHAB has access to perfectly fitted distributions (light, positive values). Furthermore, several dark bins (negative values) indicate higher acceptance of larger requests, when uniform distributions are used, e.g., $B = 2$ and $S = 4$.

In conclusion, AHAB's performance seems to be robust against small deviations in the request generation process. But the acceptance pattern changes. Larger deviations are unlikely given today's estimation methodologies but would require a more extensive analysis of AHAB's behavior.

H. How should we design the cluster?

Finally, this section evaluates how the results are affected by the host capacities. Fig. 9 illustrates the CU utilization of the different algorithms for varying computation ($\hat{C}$) and network ($\hat{B}$) capacities of the hosts. The arrival rate is $\lambda = 12$ for all setups with $\hat{C} = 8$ and $\lambda = 20$ for all setups with $\hat{C} = 16$ to keep the offered load similar. In both cases, the system is overloaded and $> 15\%$ of the requests are rejected. The leftmost group of confidence intervals shows the results for $\hat{C} = 8, \hat{B} = 8$ which are already evaluated in Sec. V-B. We recall the significant dominance of AHAB(KRAKEN). When the capacity of the hosts' up-link is doubled ($\hat{C} = 8, \hat{B} = 16$), a first observation is that the utilization increases for all algorithms. However, OKTOPUS, KRAKEN and TETRIS close the gap to AHAB. The performance difference is only ≤ 0.05 compared to ≈ 0.1 in the previous case. Moreover, OKTOPUS, KRAKEN and TETRIS perform now similar as AHAB(KRAKEN) with $\hat{B} = 8$ but at the cost of doubling the physical link capacity. This observation implies that the high utilization of the host link limits the embedding of VCs and leads to fragmented computation resources. With the increased up-link capacity, the resource ratio of a host is now $\frac{\hat{C}}{\hat{B}} = 0.5 < 1$, which is similar to the ratio of the requests that are preferably picked by KRAKEN and TETRIS (see Sec. V-D). Furthermore, a single VM can no longer block an entire host's link. This increases the probability of multiple allocated VMs at one host and reduces the fragmentation of computational resources. A second point is that TETRIS performs worse than OKTOPUS and KRAKEN. Thus, with sufficient network resources available, the benefit of mapping communication intensive with computation intensive requests vanishes.

Doubling $\hat{C}$ while keeping $\hat{B} = 8$, leads in total to lower utilization for all algorithms. In particular, the gap between OKTOPUS and KRAKEN grows as OKTOPUS embeds less efficiently and wastes more resources on the hosts' up-links.

The results for the case $\hat{C} = 16$, $\hat{B} = 16$ show that again the bandwidth is the limiting factor and one main reason why AHAB performs better than the algorithms without admission control. For this case, the utilization obtained with TETRIS and AHAB is almost the same and also OKTOPUS and KRAKEN close the gap to AHAB. Thus, the advantage of AHAB diminishes when the maximum size of the VMs decreases in comparison to the available CUs on a host and the trade-off between performance gain and computational overhead has to be done more carefully. However, further

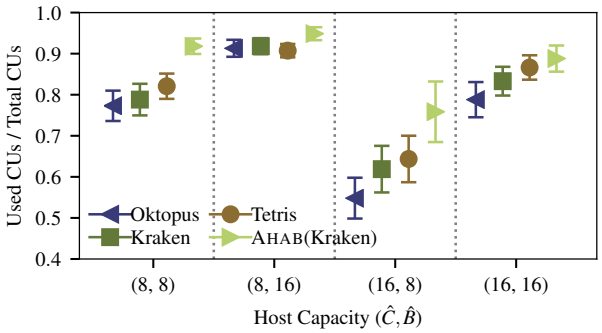


Fig. 9. Performance comparison between embedding algorithms using 95% confidence intervals of CU utilization against host capacities. Note, that the arrival rate for $\hat{C} = 8$ is 12, while it is 20 for $\hat{C} = 16$.

evaluations are necessary to analyze this more in detail.

VI. Conclusions

Virtual clusters are one of the most prominent abstractions that guarantee network performance and isolation in batch-processing and cloud computing. Their efficient embedding on the physical topology is crucial for the economical operation of such systems. This work presented TETRIS, a new VC embedding algorithm that sacrifices the embedding efficiency of a single request in order to maximize the reward in the long run. TETRIS tries to balance the utilization of resources along different dimensions by mapping together computation and communication intensive requests. The evaluations show that this approach beats algorithms such as OKTOPUS or KRAKEN.

As a second step to increase the performance of cluster embedding over time, this work proposed AHAB, a data-driven approach to admission control for VC embedding. AHAB is based on the idea of looking into the future and evaluating the benefit of the current embedding using knowledge about the distributions of the requests' attributes. AHAB shows better performance than algorithms without or with only very simple admission control. This improvement comes at the cost of higher computational complexity which however, can be controlled by AHAB's parametrization. Furthermore, the evaluation shows that the performance difference is impacted by the size of the substrate network.

In future research, it is interesting to look into the possibility provided by machine learning to reduce the online computational effort of AHAB by learning from experience as in [19], [20]. Additionally, within the prediction sequences that AHAB runs, no admission control is applied. The use of more sophisticated policies, as provided by deep learning, can enhance the decision quality of AHAB. Furthermore, we believe that cluster planning that integrates algorithm behaviors, application specifications and demands is another interesting angle for future investigation.

Acknowledgment

This work is part of a project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets).

References

[1] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM CCR*, vol. 42, no. 5, pp. 44–48, 2012.

[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM 2011*, Toronto, Ontario, Canada, 2011, pp. 242–253.

[3] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. CoNEXT 2010*, Philadelphia, USA, 2010, pp. 15:1–15:12.

[4] K. C. Webb, A. Roy, K. Yocum, and A. C. Snoeren, "Blender: Upgrading tenant-based data center networking," in *2014 ACM/IEEE ANCS*, Los Angeles, CA, USA, 2014, pp. 65–75.

[5] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. 3rd Conference on I/O Virtualization*, Portland, OR, USA, 2011, pp. 1–8.

[6] D. Li, J. Zhu, J. Wu, J. Guan, and Y. Zhang, "Guaranteeing Heterogeneous Bandwidth Demand in Multitenant Data Center Networks," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1648–1660, 2015.

[7] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-varying Network Reservations in Data Centers," in *Proc. ACM SIGCOMM 2012*, vol. 42, Helsinki, Finland, 2012, pp. 199–210.

[8] M. Rost, C. Fuerst, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," *ACM SIGCOMM CCR*, vol. 45, no. 3, pp. 12–18, 2015.

[9] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Online and Elastic Resource Reservations for Cloud Datacenters," *IEEE/ACM Trans. Netw.*, vol. PP, no. 99, pp. 1–14, 2017.

[10] R. Yu, G. Xue, X. Zhang, and D. Li, "Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers," in *Proc. IEEE INFOCOM 2017*, Atlanta, GA, USA, 2017, pp. 1–9.

[11] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in *Proc. SOSP '17*, Shanghai, China, 2017, pp. 153–167.

[12] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Unleashing the potential of data-driven networking," in *Proc. COMSNET 2017*, Bengaluru, India, 2017, pp. 1–8.

[13] L. Yu and H. Shen, "Bandwidth Guarantee under Demand Uncertainty in Multi-tenant Clouds," in *Proc. IEEE ICDCS 2014*, Madrid, Spain, 2014, pp. 258–267.

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *Proc. 8th NSDI*, vol. 11, Boston, MA, USA, 2011, pp. 295–308.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM 2008*, vol. 38, Seattle, WA, USA, 2008, pp. 63–74.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," vol. 4, no. 1, pp. 1–49, 2012.

[18] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Sci. Technol. Cent. Cloud Comput. Tech Rep*, vol. 84, 2012.

[19] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *Proc. 12th CNSM*, Montreal, Canada, 2016, pp. 10–18.

[20] A. Blenk, P. Kalmbach, W. Kellerer, and S. Schmid, "O'zapft is: Tap Your Network Algorithm's Big Data!" in *Proc. ACM Big-DAMA*, Los Angeles, CA, USA, 2017, pp. 19–24.

Controlling software router resource sharing by fair packet dropping

Vamsi Addanki, Leonardo Linguaglossa, James Roberts and Dario Rossi
Telecom ParisTech, Paris, France – first.last@telecom-paristech.fr

***Abstract*—The paper discusses resource sharing in a software router where both bandwidth and CPU may be bottlenecks. We propose a novel fair dropping algorithm to realize per-flow max-min fair sharing of these resources. The algorithm is compatible with features like batch I/O and batch processing that tend to make classical scheduling impractical. We describe an implementation using Vector Packet Processing, part of the Linux Foundation FD.io project. Preliminary experimental results prove the efficiency of the algorithm in controlling bandwidth and CPU sharing at high speed. Performance in dynamic traffic is evaluated using analysis and simulation, demonstrating that the proposed approach is both effective and scalable.**

I. Introduction

Controlling how bandwidth is shared between concurrent flows is a classical issue in networking. While there are multiple objectives in this field and many proposed mechanisms, we concentrate in this paper on max-min fair sharing between a dynamically changing population of flows in progress.

The advantages of imposing bandwidth fairness have been repeatedly discussed since Nagle's pioneering observations [22]. See [23, Sec. 7] for a very clear summary. Satisfactory performance is maintained even when end-systems do not comply with TCP-like congestion control. More efficient high speed transport protocols can be introduced without requiring them to be friendly to legacy TCP. Implicit service differentiation is realized in that low rate streaming flows naturally experience negligible packet loss and delay.

In emerging high-speed software routers, flow throughput may additionally be impeded by resources other than bandwidth. We concentrate in this paper on the CPU executing virtualized network functions for packet forwarding and processing. CPU capacity is measured in cycle/s and flows may differ widely in their per-packet requirements depending on the functions they execute. The considered objective here is max-min fair flow rates in cycle/s, the product of the packet/s rate and the number of cycles needed to process each packet.

In Ghodsi *et al.* [12], fair bit/s bandwidth sharing and fair cycle/s CPU sharing are coupled in the notion of dominant resource fairness (DRF). In this work, we propose rather to control fair sharing of bandwidth and CPU resources independently (i.e., without using weights that depend on the dominant resource). This is both simpler to implement than DRF and fulfils a multi-resource sharing objective that is in significant ways preferable [8].

The mechanism envisaged in [12] and [8] for imposing fair shares is a scheduler like start time fair queuing (STFQ) [13]. However, this approach is hardly compatible with the hardware and software optimizations that are necessary to keep up with line speeds of 10 Gbps and more on a single CPU core. These optimizations notably require packets to be batched for both I/O and processing making implementation of classical scheduling algorithms problematic if not impossible, as argued in [28]. We therefore propose a more flexible software oriented solution based on fair packet dropping.

A number of approximate fair dropping algorithms have already been proposed for fair bandwidth sharing, such as FRED [17], CHOKe [24], RED-PD [21] and AFD [23]. In a preliminary evaluation, we found these algorithms imprecise and difficult to implement, especially in the present context of a software router. We have preferred to explore an original *exact* fair dropping algorithm. This algorithm is shown to be scalable since it operates only on the limited number of flows that would currently be backlogged in a fair queuing scheduler [18].

Despite strong current interest in network function virtualization, there is still little published work on how one might control CPU sharing between concurrent flows. A recent paper by Vasilescu *et al.* recognizes the need for fair sharing and advocates a differential congestion marking scheme to account for flows with different cycle/packet costs [29]. Shin *et al.* have previously advocated a similar congestion marking scheme [26]. Marking is less robust than fair dropping since, to achieve fairness, it is necessary that end-systems respond correctly to the congestion notification. In environments where this is a reasonable assumption, our proposed algorithm could be trivially modified to perform accurate fair marking instead of fair dropping.

Our main contributions here are to define an original algorithm to control CPU sharing using fair dropping and to evaluate its performance by analysis, simulation and experimentation. Application of the same approach to control bandwidth sharing is also novel but does not differ radically from earlier proposals. Successive presentation of algorithms and results for bandwidth sharing and CPU sharing usefully highlights the additional complexity in controlling the latter.

We first discuss salient features of software routers (Sec.II) before introducing the proposed fair dropping algorithms and illustrating their behavior by simulation (Sec. III). A prototype implementation in the FD.io software router is

ISBN 978-3-903176-08-9

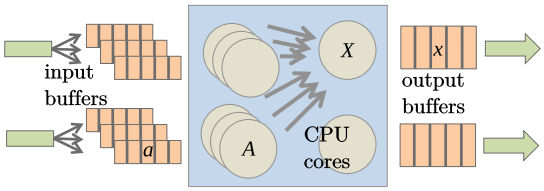


Fig. 1. Sketch of software router: core A forwards flows from input buffer a, core X handles all packets destined to output x.

then described and experimentally evaluated (Sec.IV). Finally, simulations in dynamic traffic confirm the scalability of the proposed approach and demonstrate its throughput performance (Sec. V).

II. Software routers

We highlight features of emerging high-speed software routers that are significant for controlled resource sharing. For a more complete discussion, see [1], [5], [11], for instance.

A. Throughput bottlenecks

Flow throughput may be momentarily impeded by multiple bottlenecks in a software router. We consider here just two, output link bandwidth and CPU forwarding capacity. Note that CPU forwarding includes basic forwarding operations but also more complex tasks like encryption or a range of virtualized network functions.

Controlling bandwidth sharing by scheduling and buffer management is a classical function in networking with many proposed solutions. In software routers a notable example is the DPDK QoS framework that includes a variety of mechanisms ranging from the token bucket to a hierarchical weighted fair queuing scheduler [2]. These mechanisms are necessarily implemented in a CPU core that sees all packets destined for a given output port. In Fig. 1, the core in question for output port x is labelled X. The fair dropping algorithm we propose would similarly be implemented in core X (of course, the same core could process multiple outputs). It is an efficient alternative to scheduling.

The flexibility of software routers means packet processing capacity can be more closely matched to demand than in a hardware router and CPU can be a bottleneck. In Fig. 1 core A handles flows from input buffer a but, depending on demand, it might handle more buffers from multiple NICs. A CPU core becomes a bottleneck when flows emit packets too fast yielding a compute load greater than capacity and leading therefore to packet drops. In the absence of any control, the most aggressive flows will seize more CPU cycle/s than the others. Unfairness is exacerbated by the unequal per-packet costs of flows with different protocols [1] or middlebox function requirements [12]. We demonstrate that a lightweight fair dropping mechanism ensures flows get their fair share of CPU cycles. Implementing a scheduler like DRR [27] or STFQ [13] in this context, on the other hand, appears problematic to say the least.

B. Flow-awareness

High-speed software routers are intrinsically flow-aware. Flow-awareness is facilitated by NICs implementing receive side scaling (RSS). RSS performs a hash of packet header fields (e.g., the usual 5-tuple) and maps this to distinct queues, mainly for the purpose of load balancing over multiple CPU cores. Individual threads of packet processing applications are bound to a CPU core and, using kernel-bypass stacks (such as DPDK [3], netmap [25], PF_RING [10] or pfq [9]), threads consume independent streams of packets, each from a different RSS queue. In Fig. 1, the splitting of incoming traffic over multiple input buffers is flow-aware. Importantly for the implementation of flow-aware functionality, the hash is recorded in packet metadata and can be accessed by router software.

A significant advantage of flow-awareness in a software router is that all packets of the same flow are processed by the same core bringing efficiencies and enabling limited per-flow state. FIB lookup efficiency is enhanced, for instance, since only the first packet of a flow will typically require a RAM memory access while the result will remain in cache for subsequent packets. Flow state is necessary for mechanisms like the present fair dropping proposal.

C. Batch-mode processing

It is significant that high-speed software routers and their NICs generally deal with packets in *batches* rather than individually. This is a necessary optimization for line-speed I/O and greatly improves the efficiency of packet processing. In kernel bypass stacks, batching significantly reduces interrupt pressure compared to per-packet operation. The CPU handling an input buffer therefore typically polls for available packets. It visits the buffer, grabs all waiting packets up to a maximum number and processes the entire batch before returning to grab another batch.

The most recent high-speed software routers also perform forwarding tasks successively on batches of packets [16], [5], [1]. Each task in the processing graph is performed on all packets in the batch before the CPU moves to the next task. Batched processing optimizes the use of the CPU instruction cache as code for a task only needs to be fetched once for the entire batch. In addition, the overhead of managing graph traversal (counters, pointers, calls,...) is minimized since most operations need be performed once only for the entire batch. Processing efficiency improves with the size of the batch and mechanisms beyond simple polling are generally employed to ensure the batch is large enough. Batching makes it impossible to implement classical schedulers that rely on dequeue operations being triggered by individual packet departures [28].

III. Fair dropping

We present the fair dropping algorithm and numerically illustrate how it realizes per-flow fair bandwidth and CPU sharing under static demand.

A. Fair rates by dropping

Suppose packets are handled simultaneously by two service systems, one the actual buffer management system implemented in the router (e.g., FIFO), the other a shadow system implementing a more sophisticated scheduler (e.g., per-flow FQ). Packets that are dropped in one system are also dropped by the other so that both systems yield exactly the same rate over the lifetime of a flow.

The shadow system in our proposal is virtual and makes dropping decisions based on a measure of per-flow virtual queue occupancy. This measure is depleted between packet arrivals, at a rate that varies depending on the number of active flows, and incremented by packet length on the arrival of every batch. If we correctly track the virtual queue occupancies at arrival instants, and make drop decisions in the shadow system aggressively enough to avoid additional drops due to buffer overflow in the real system, flow rates are entirely determined by the shadow system. In particular, if the shadow system implements per-flow head-of-line processor sharing, the long-term flow rates will be max-min fair.

Fair dropping, as a software solution, is inherently more flexible than scheduling. In particular, the shadow system can be readily turned off when not needed, economizing CPU usage. If the sum of rates of flows in progress is currently less than the capacity of the resource in question, there is no need to impose fairness. This may be the usual case (e.g., for a backbone link, or a CPU that only performs forwarding) with fair dropping ready to be turned on as necessary (e.g., when a high rate server-to-server flow starts up, or a new IPSec flow suddenly saturates the CPU).

B. The algorithms

Algorithm 1 is used for fairly sharing either bandwidth or CPU capacity. It uses a table called ActiveList, containing the current virtual queue size ($flow.vq$) for all backlogged flows (i.e., $flow.vq > 0$) indexed by identifiers $flow$ (typically a hash of header fields). Following the arrival of a batch of packets, ActiveList is updated using lines 3 to 15. Virtual queue occupancies $flow.vq$ are reduced by their max-min fair share of service capacity accumulated since the last call. If $flow.vq$ goes to zero, the flow is removed from ActiveList.

Lines 16 to 28 deal with the newly arrived packet or packets. Packets are dropped if the virtual queue of their flow exceeds a threshold θ. New flows are added to ActiveList and virtual queues are incremented by the size of the new packet. For bandwidth sharing $packet.length$ is measured in bytes while for CPU sharing it is an estimate of the number of cycles needed to process the packet.

For bandwidth sharing Algorithm 1 is sufficient and must be performed in a CPU receiving all packets destined to the considered output link. For CPU sharing, it is necessary to perform additional instructions to account for the fact that $packet.length$, the number of cycles needed to process the packet is not known *a priori*. Moreover, the number of cycles used to process a batch includes an overhead accounting for the cycles expended on dropped packets.

Algorithm 1 Virtual queue updates and dropping performed on arrival of a batch of packets.

1: **Given**: Δt - time since last update, C - service rate, $\mathcal{B}$ - ActiveList of backlogged flows, $\mathcal{P}$ - batch of new packets, θ - a threshold.
2: input $\mathcal{P}$
3: $credit = C\Delta t$
4: **while** $credit > 0$ **and** $|\mathcal{B}| > 0$ **do**
5: $\quad share = credit/|\mathcal{B}|$
6: $\quad credit = 0$
7: $\quad$ **for** each $flow \in \mathcal{B}$ **do**
8: $\quad\quad$ **if** $share < flow.vq$ **then**
9: $\quad\quad\quad flow.vq \mathrel{-}= share$
10: $\quad\quad$ **else**
11: $\quad\quad\quad credit \mathrel{+}= share - flow.vq$
12: $\quad\quad\quad \mathcal{B} = \mathcal{B} \setminus flow$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end while**
16: **for** each $packet \in \mathcal{P}$ **do**
17: $\quad$ **if** $flow(packet) \in \mathcal{B}$ **then**
18: $\quad\quad$ **if** $flow.vq > \theta$ **then**
19: $\quad\quad\quad$ drop $packet$
20: $\quad\quad\quad \mathcal{P} = \mathcal{P} \setminus packet$
21: $\quad\quad$ **else**
22: $\quad\quad\quad flow.vq \mathrel{+}= packet.length$
23: $\quad\quad$ **end if**
24: $\quad$ **else**
25: $\quad\quad \mathcal{B} = \mathcal{B} \cup flow$
26: $\quad\quad flow.vq = packet.length$
27: $\quad$ **end if**
28: **end for**

Algorithm 2 Cost calculation for CPU sharing for flows of different types.

1: **Given**: $\mathcal{P}$ - batch of packets, Δc - cycles consumed to process $\mathcal{P}$, w_k - relative weight of type k
2: $sumw = \sum_{packet \in \mathcal{P}} w_{type(packet)}$
3: **for** each $packet \in \mathcal{P}$ **do**
4: $\quad packet.cost = (w_{type(packet)}/sumw)\Delta c$
5: $\quad flow.vq = flow.vq + packet.cost - packet.length$
6: **end for**

Algorithm 2 apportions the measured overall batch processing cost Δc in proportion to weights giving the relative number of cycles needed to process packets of given type (e.g., a packet needing to consult an ACL may need more than ten times as many cycles as a packet that is simply forwarded). This cost calculation can only be performed after processing is complete while Algorithm 1 uses an assumed $packet.length$ to make drop decisions. The latter might be an average cost estimate derived by prior measurement or a real time updated average based on $packet.cost$ estimated by Algorithm 2 for packets of the given type for previous

TABLE I
BANDWIDTH SHARING

		Utilization breakdown	Latency
FQ	TD	0.59 / 0.35 / 0.06	47.2 / 3.54 / 2.03
FIFO	TD	0.59 / 0.35 / 0.06	29.2 / 29.1 / 28.9
FQ	FD	0.45 / 0.45 / 0.10	22.1 / 19.6 / 2.70
FIFO	FD	0.45 / 0.45 / 0.10	18.9 / 18.9 / 19.4

batches. The virtual queue lengths must be corrected (line 5) to ensure they accurately track the actual numbers of expended cycles. Algorithm 2 can be performed at the start of a new processing cycle, before Algorithm 1, using data for the packets $\mathcal{P}$ processed in the previous cycle. Δc is the total number of cycles consumed between successive polling events.

Algorithm 1 realizes per-flow max-min fairness. It could easily be extended to realize more general objectives like hierarchical weighted fairness [7], for instance. Parameters identifying flow classes and weights would be stored in the flow table along with current virtual queue lengths. These would be used to derive flow specific shares in place of the common value computed in line 5.

C. Scalability

It is commonly believed that per-flow fair scheduling is not scalable since compute time grows with the number of flows and this number can attain many thousands on high speed links. This reasoning would apply similarly to Algorithms 1 and 2. In fact, the algorithms *are* scalable since, though the number of flows *in progress* may be very large, the set of *active* flows $\mathcal{B}$ includes only those that currently have a backlog. Under reasonable assumptions about the stochastic process of flow arrivals this number is small with high probability even for very high speed links, as demonstrated analytically and by trace driven simulation in [18].

The underlying analytical model is a processor sharing (PS) system with Poisson flow arrivals that has simple and robust performance characteristics [6]. Let ρ denote the PS server load, $\rho =$ flow arrival rate $\times$ flow size / server capacity, with $\rho < 1$ for stability. The number of active flows has a geometric distribution (i.e., $\mathbb{P}[\text{active flows} \geq x] = \rho^x$) and the expected completion time of a flow of size s is $s/(1-\rho)$. As a measure of flow throughput we use the reciprocal of the normalized flow completion time, $(1-\rho)$. These results apply for a wide range of stochastic demand models, as discussed in [6]. In particular, they do not depend on any assumption about the distribution of flow size.

Scalability in the present context is demonstrated by simulation in Sec. V while in this and the following section we illustrate algorithm performance for static sets of concurrent flows.

D. Bandwidth sharing

For bandwidth sharing, fair dropping can be implemented before buffering packets at the output port. Bandwidth sharing is then as fair as that realized with fair queuing schedulers like DRR [27] or STFQ [13].

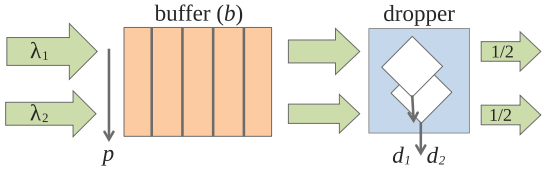


Fig. 2. Dropper for CPU sharing is placed after the buffer but uses times packets arrive at the buffer.

Table I illustrates through simulation results for a toy example, that FQ scheduling is *not necessary* for fairness while FD is *essential*. Three flows emit unit size packets as a Poisson process at respective rates 1, 0.6 and 0.1 and share a unit rate link using FQ or FIFO. Excess packets suffer tail drop (TD) in the buffer preceding the link, or fair drop (FD) using Algorithm 1. ActiveList updates are performed here after every packet arrival. Buffer capacity is 30 packets and, for fair dropping, we set threshold θ to 10. Latency is measured in packet transmission times.

The results confirm that FD is sufficient for fair throughput while significantly increasing the latency of the low rate third flow. The negative impact of higher latency, for VoIP applications say, can be removed by replacing the link FIFO by a priority scheduler where packets belonging to flows absent from ActiveList on their arrival are served first [19], [14]. Latency could also be reduced by operating the drop algorithm with a rate C somewhat less than the link rate (e.g., as in [4]).

E. CPU sharing

To share CPU capacity, dropping can only take place after the input buffer. To emulate a fair scheduler, the shadow system must however make drop decisions based on packet arrival times at the buffer as recorded in a time stamp. These 'fair drops', based on the size of the per-flow virtual queues, are in addition to any 'tail drops' due to buffer saturation. The relative proportions of tail drops and fair drops depends on the choice of threshold θ. It is also necessary here to account for the fact that the act of dropping a packet consumes CPU cycles that are otherwise to be shared fairly.

a) An adaptive threshold: To illustrate how fair dropping realizes fair CPU sharing, consider the toy example of Fig. 2. N flows ($N = 2$ in the figure) bring processing requirements λ_i (packet/s $\times$ cycle/packet) and are served by a unit capacity CPU. A fraction p of each flow is lost due to buffer overflow. The remaining $(1-p)$ fractions of each flow share the CPU in max-min fashion, thanks to fair dropping, and suffer additional drops at rates d_i.

This queuing system is particularly complicated and we have no analytic results to determine the impact on p and the d_i of particular choices of buffer size b and drop threshold θ. Note, however, that while b is system dependent and fixed, θ is simply a program parameter and can be set and reset as

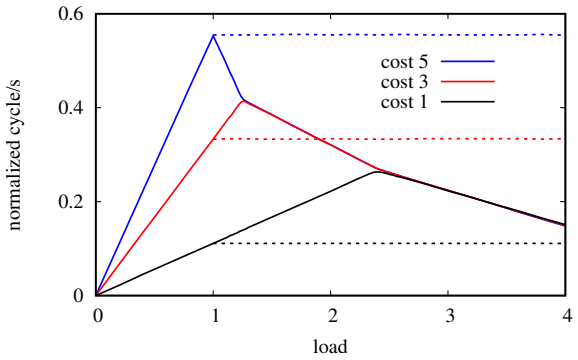


Fig. 3. CPU flow throughput as a function of load for 3 flows with equal packet/s rates and respective CPU costs 5, 3 and 1 and drop overhead 0.5; dotted lines show throughput without fair dropping.

necessary. It is possible, in particular, to adapt θ from one batch to the next based on observations of a performance objective.

A first performance objective is to make the probability of buffer saturation negligibly small. This is necessary to avoid undue loss for flows emitting at a rate less than the fair rate or flows consisting of a single packet like DNS queries. A second objective is to maintain processing efficiency by making batches as large as possible (cf. Sec. II-C). We therefore adapt θ as follows: if the polled vector is maximal (suggesting impending saturation), multiply θ by a factor $\alpha < 1$ to induce more fair drops; if the polled vector is not maximal, multiply θ by $\beta > 1$. The choice of parameters α and β is not highly critical. In our simulations, setting $\alpha = .5$ and $\beta = 1.2$ gave satisfactory results.

b) Dropping overhead: Any dropped packet consumes cycles and this overhead reduces flow throughput. The overhead includes the cycles used to run Algorithms 1 and 2 but is mainly due to the cost of bringing packets into the CPU, as is necessary to determine their flow identity.

Fig. 3 presents per-flow cycle/s throughput for a toy system with 3 flows, each emitting packets as a Poisson process at the same rate, as a function of load. The flows have different relative per-packet processing requirements: flow 1 packets cost 5 units, flow 2 packets 3 and flow 3 packets 1 (the value of the cost unit in compute cycles is not significant, only their ratio matters). Dropped packets consume 0.5 units of CPU. Offered load on the x axis is the sum of the products (packet rate $\times$ cost) divided by the CPU capacity (i.e., $\sum \lambda_i$).

The figure plots normalized cycle/s throughputs (the sum of throughputs is 1 at load 1) against load. Dotted lines show throughputs realized in the absence of fair dropping. These results confirm that fair dropping realizes max-min fairness, e.g., the cost 1 flow suffers no loss until its input rate exceeds the fair rate when all flows have the same throughput. On the other hand, the sum of throughputs decreases with increasing load due to the cost of dropping. Throughputs go to zero at load 6 when the CPU is entirely busy dropping packets. Throughput can be bounded away from zero by setting a minimum threshold at the cost, however, of significant loss for low rate and single-packet flows.

IV. Implementation

We have implemented the fair dropping algorithms on a real software router. In the following we describe the testbed, outline the software architecture and present experimental results.

A. Experimental setup

Algorithms 1 and 2 run in the Vector Packet Processing (VPP) software router that is part of the Linux Foundation's FD.io project [1]. The VPP router is deployed in a server platform based on two Intel Xeon E52690 processors, each with 24 cores running at 2.60 GHz, equipped with two 10 Gbps Intel X520 NICs directly connected with SFP+ interfaces. The two processors and two line cards are isolated to logically create two independent nodes. This is realized using non-uniform memory access, to make portions of RAM accessible to only one line card, and CPU core binding where particular cores are mapped to a specific process and interrupts are deactivated.

Of the two nodes, one acts as traffic generator and sink (TGS) while the other is the system under test (SUT), the software router equipped with our FD algorithms. The TGS continuously sends a stream of packets to the SUT which processes them and sends them back to the TGS. We can measure the throughput of the SUT as the return input rate to the TGS. To validate the FD algorithm and its scalability, the TGS sends traffic consisting of 64-byte packets at 10 Gbps (corresponding to an input rate of 14.88 Mpps). IP addresses and port numbers are set to emulate a number of distinct constant rate flows. To stress the system, the FD algorithms are executed on a single core handling all traffic.

B. Software architecture

Our implementation[1] works on bandwidth and CPU sharing. The fair dropper is currently implemented within an FD.io node [1], but could alternatively be implemented as a lower-level primitive of the DPDK QoS framework [2]. Porting Algorithm 1 to DPDK is straightforward and would be sufficient for bandwidth sharing. However, per-packet cost estimates derived in Algorithm 2, as necessary for CPU sharing, must be made available to DPDK and this requires further work. Thus, while the experimental results presented below are specific to the FD.io implementation, they may be considered as a more general proof-of-concept for a software line-rate implementation of fair dropping.

Vector Packet Processing (VPP) [20], is a kernel-bypass application that *reads* and *processes* packets in batches. VPP consists of a set of software functions that logically abstract network operations of different layers of the protocol stack (examples of functions are `l2_input` or `ip4_lookup`). VPP links such functions to form a *processing graph* and

[1] https://github.com/TeamRossi/vpp-bench

each function (represented by a node in the graph) is applied to packets as necessary in order to implement packet processing and forwarding. Batched reads are performed by DPDK drivers accessing NIC hardware and significantly reduce interrupt pressure (I/O batching). Processing is also performed in batches of packets – called *vectors* – optimizing usage of the underlying CPU architecture. In other words, during graph traversal, each node function is applied to a full vector of packets (compute batching) before continuing to the next node. For bandwidth sharing, Algorithm 1 runs in the final output nodes of the VPP processing graph while, for CPU sharing, both algorithms are implemented in the initial node called `dpdk-input`.

In our implementation, FD operations are performed once per batch thus matching the typical work-flow of a VPP router. This reduces the induced overhead without unduly impacting realized fairness (cf. Sec. III). To reduce computational complexity, we identify the flow using the 5-tuple hash computed by the NIC for RSS queues. This is accessible via the `hash.rss` variable within the `mbuf` DPDK structure.

Flows are stored in a flow table. The data structure used is a hash-table with 4K rows each receiving up to 4 24-byte flow records. The row is addressed by 12 bits of the RSS hash and the flow record uses 8 more bits of the hash to distinguish up to 4 flows mapped to the same row. The row size is aligned to fit two cache lines in our platform. In view of the scalability results discussed in Sec. III-C, the flow table size is largely sufficient to ensure the probability of misidentifying a flow is negligible.

We additionally maintain a separate data structure identifying the *active flows*, that is, the flows that currently have a positive virtual queue. This structure has two roles: it identifies the small set of flows to which the FD algorithm must be applied, and it enables data for this set to be maintained in CPU L1 or L2 cache. Flow state in the present implementation is confined to the flow identifier and the current virtual queue length. However, space remains for additional state needed by more complex objectives, like hierarchical weighted fairness for instance.

Implementation of Algorithm 1 for CPU sharing requires access to packet arrival times using a timestamp. NIC time stamping is currently available through the DPDK `rxtx_callback` function. When the callback is executed, the Time Stamp Counter (TSC) is accessed[2] and the TSC register value is written to the DPDK `mbuf` user data field `udata64`.

C. Bandwidth sharing

To illustrate FD induced bandwidth sharing we generate a workload consisting of 20 flows with progressively decreasing arrival rates: the flow with rank 1 has an arrival rate 10 times higher than that of flow 20. We produce a bottleneck by rate limiting the output port to a fraction α of the input rate.

[2]TSC is a 64-bit register whose purpose is to count the CPU clock cycles

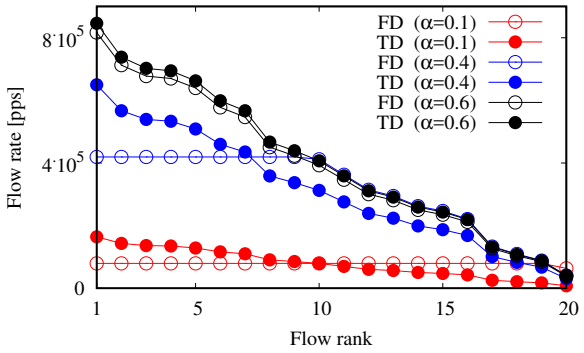


Fig. 4. Performance of bandwidth sharing: experiments for different output rates $\alpha \times 10$ Gbps (10Gbps line card, 1 CPU core, IPv4 forwarding, skewed packet rates).

Fig. 4 presents experimental results for 3 values of α. Per-flow output rates are plotted with either FD or TD (i.e., without differential dropping). With $\alpha = 0.1$, all flows are able to attain the fair rate under FD while rates are proportional to input rates under TD. With $\alpha = 0.4$ the available bandwidth increases and FD affects only those flows (of rank 1 to 10) that exceed the fair rate. In both cases, the overall drop rate under TD and FD is exactly the same, only *which* packets are dropped differs.

With $\alpha = 0.6$ the output link is no longer a bottleneck and flow rates are limited by the CPU processing capacity[3]. This implies flow rates are reduced proportionally due to input buffer saturation. The difference between the top two black lines in the figure is due to the overhead of our current non-optimized implementation of the FD algorithm (which doesn't actually drop any packets in this case).

D. CPU sharing

In our experiments on CPU sharing, the TGS creates 20 equal packet rate flows belonging to one of two different types: packets of type-L flows require "light" processing, while packets of type-H flows have a "heavy" cost, consuming r times more cycles than type-L. Per-packet cost depends on the amount of processing in the VPP graph for both I/O and computation and can be readily measured using VPP primitives [20]. In the experiments, 18 type-L flows send IPv4 packets requiring standard processing: longest prefix matching and next hop forwarding. Two high-cost flows additionally pass via a busy loop whose length can be precisely controlled to modulate the ratio r of H to L type costs.

Experimental results are represented as Sankey diagrams in Figure 5 where TD and FD are compared for $r = 10$ with an input rate of 14.88 Mpps. With tail drop, 11.02 Mpps are dropped at the NIC interface, and the rest is processed by the SUT. Tail drop makes no explicit decision as to which

[3]This corresponds to about 8.5 Mpps IPv4 forwarding throughput

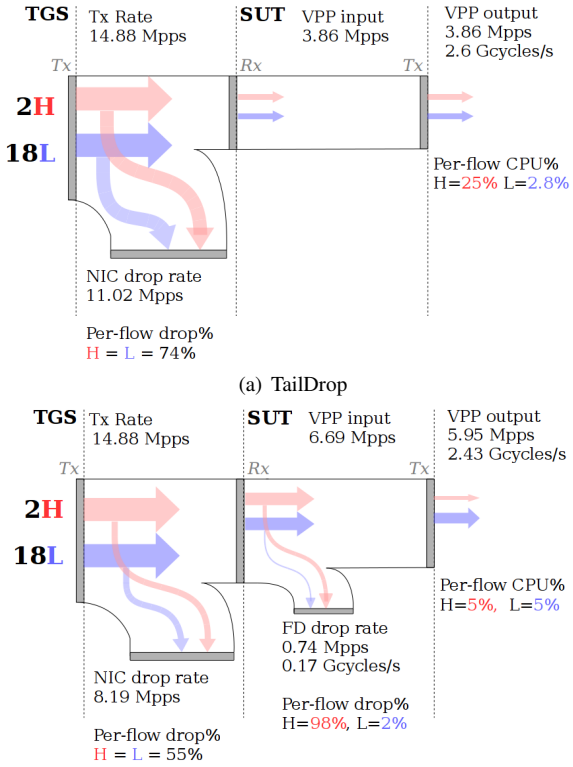


Fig. 5. Sankey diagrams for Tail Drop (a) and Fair Drop (b) experiments

packets should be dropped and, since packet rates are equal among flows, drops affect equally the H and L classes (74%). It follows that all flows have the same bit rate while each of the H flows individually consumes 25% of the CPU cycles.

FD radically changes the operational point. The NIC drops 8.19 Mpps (55%) increasing the traffic processed by the SUT to 6.69 Mpps. The FD decision consumes 0.17 G cycle/sec (the overhead of the FD algorithm) and affects 0.74 Mpps of packets. As expected, 98% of the dropped packets are of type-H. This differentiation realizes fairness in terms of CPU cycles, since each of the 20 flows now receives exactly the 5% fair share of CPU. Notice also that, in this particular scenario, the overall rate of packets forwarded by the SUT increases: throughput is 5.95 Mpps with FD, compared to 3.86 Mpps with TD).

The drop threshold θ is fixed in these experiments. The packet arrival rate is such that it is not possible to eliminate buffer saturation. With respect to the overhead due to dropping packets, we measured the following average costs. To successfully send a type-L packet requires 350 cycles while a dropped packet costs 208 cycles made up of 120 for I/O, 50 for freeing packet memory and 38 for executing the drop algorithms.

To further illustrate the difference in fairness between FD and TD we repeat the above experiment with the value of r ranging from 1 to 14. For each value we compute the Jain fairness index between types L and H for both

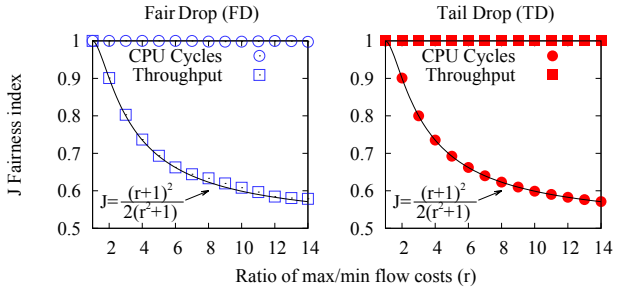


Fig. 6. Throughput and CPU fairness indices for FD (left) and TD (right) as a function of the ratio r of type-H to type-L costs.

average flow cycle/s and packet/s throughputs [15]. With 2 types and respective metrics x_L and x_H, the Jain index is $(x_L + x_H)^2/(2(x_L^2 + x_H^2))$. Results for FD and TD, with x representing cycle/s and packet/s throughputs, are shown in Figure 6. The figure confirms that FD fairly shares CPU cycles while TD is fair in terms of packets/s (only because the input rates are equal). In contrast, FD packet/s throughputs and TD cycle/s throughputs are unfair in the ratio r with index $(r+1)^2/2(r^2+1)$.

V. Performance in Dynamic Traffic

We evaluate throughput performance in dynamic traffic and demonstrate scalability by simulation.

A. Demand model

Traffic demand is modeled as a Poisson process of flows of finite size of different types. Packets have constant size in bytes. Per-packet processing cost depends on the flow type and is assumed constant for all packets of the same flow.

We distinguish full-rate flows and single-packet flows. Full-rate flows last until 30000 unit size packets are successfully transmitted. As noted in Sec. III-C, we expect performance to be insensitive to the size distribution. Constant size is chosen for faster convergence of the simulations. The stream of single-packet flows is intended to include traffic from flows emitting packets at a rate less than the typical fair rate, possibly because of other bottlenecks on their path. The packets of such flows appear to buffer management as a succession of distinct single-packet flows.

Rather than simulating a transport protocol like TCP, we suppose full-rate flows emit packets as a Poisson process. They continue emitting packets until the number of successfully transmitted packets equals the flow size. The Poisson rate may be large and constant when flows are assumed unresponsive to congestion. To represent responsive flows we assume the Poisson rate is set to a value somewhat larger (10% here) than the rate that would be realized by a hypothetical ideal transport protocol. This simplification facilitates the evaluation of salient features of the considered algorithms, as presented below.

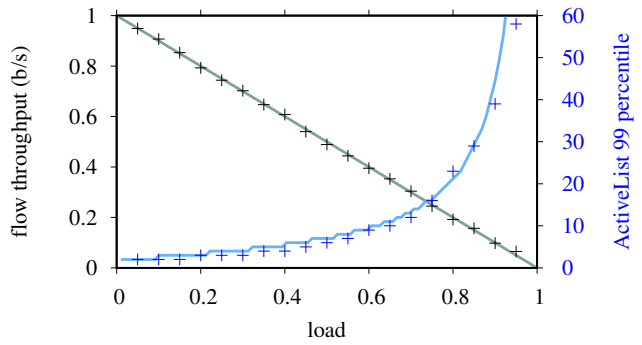


Fig. 7. Performance of bandwidth sharing: normailized throughput and ActiveList 99^{th}percentile with per packet updates; lines plot analytical results $(1-\rho)$ and $\lceil -2/\log_{10}\rho \rceil$.

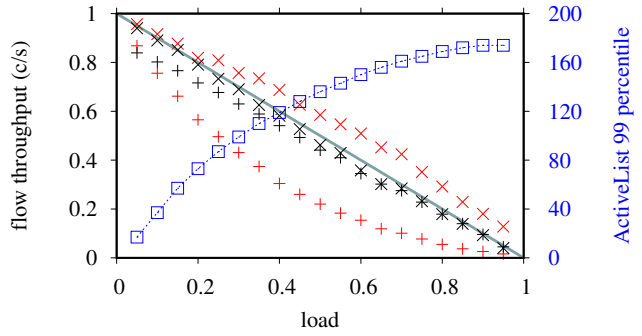


Fig. 8. Performance of CPU sharing: normalized cycle/s throughput with fair dropping (black) and tail dropping (red) for 2 flow classes, class 2 ($\times$) requires 10 times more cycles/packet than class 1(+), relative cost of dropped packets 0.5; ActiveList 99^{th}percentile for fair dropping (blue $\square$).

B. Bandwidth bottleneck

Let λ_f be the arrival rate of full-rate flows and λ_s the arrival rate of single-packet flows. The load of the unit capacity link is then $\rho = 30000\lambda_f + \lambda_s$. Given ρ, λ_f and λ_s are set so that full-rate flows contribute 80% of link load while single-packet flows make up the remaining 20%. Full-rate flows are unresponsive and emit packets at a constant rate such that total instantaneous demand largely exceeds link capacity.

Fig. 7 plots throughput (left y-axis) and the 99^{th}percentile of the distribution of the number of flows in ActiveList (right y-axis) as functions of load ρ (flow arrival rate $\times$ mean flow size / link capacity). The measure of throughput is the ratio of average flow size to average flow duration (equal to $1-\rho$ for processor sharing, as discussed in Sec. III-C). The 99^{th} percentile of the PS model is $\lceil -2/\log_{10}\rho \rceil$. The close agreement between analysis and simulation confirms that fair dropping is an effective control for bandwidth sharing even when flows are unresponsive.

C. CPU bottleneck

To evaluate the effectiveness of fair dropping in sharing a CPU bottleneck we simulate a mix of single-packet flows with unit per-packet cost and two types of full-rate flows with respective per-packet costs 1 and 10. The relative cost of dropping a packet of any type is 0.5. Single-packet flows contribute 20% of load while the full-rate flow types each contribute 40%. The buffer size is 512 packets and maximum batch size is 256. Fair dropping threshold θ is adaptive between 20 and 50000 cost units using multipliers $\alpha = .5$ and $\beta = 1.2$ (cf. Sec. III-E).

If the cost of dropping were null, our simulation results (not shown here) confirm that, as for bandwidth sharing, fair dropping yields a common cycle/s flow throughput equal to $1-\rho$, even when all flows are unresponsive. Unfortunately, this is not the case when the drop overhead is not negligible. In dynamic traffic, at some point the number of active flows will attain a level at which the CPU is saturated even when all packets are dropped (cf. Sec. III-E). Flow throughput then goes to zero and cannot recover since flows in progress do not complete while new flows continue to arrive. The impact can be mitigated by imposing a minimum threshold θ but at the cost of significant tail drops affecting single-packet flows.

It is important to note that this instability would occur with any active queue management or scheduling algorithm that selectively drops packets within the CPU. To effectively control unresponsive flows it would be necessary to selectively discard packets *before* they are polled by the CPU. We intend in future work to investigate the possibility of piloting such a mechanism using the fair dropping algorithm to identify the unresponsive flows in question.

Fair dropping remains an effective means for controlling CPU sharing between *responsive* flows with different per-packet costs. When fair dropping is employed, we know concurrent flows are allocated the same cycle/s throughput. We therefore assume responsive flows emit packets at a rate such that their cycle/s rate (packet/s $\times$ cost/packet) is 10% greater than the current fair rate. When fair dropping is absent, all packet loss is through tail drop and concurrent flows experience the same drop rate. As flow packet rate is determined by this drop rate (e.g., by TCP congestion control), we therefore derive a common packet/s rate for flows such that the sum of cycle/s rates is 10% greater than capacity. The 10% excess is meant to approximately capture the impact of a transport protocol like TCP that progressively increases flow rate until drops occur.

Fig. 8 plots throughput on the left y-axis, for tail dropping (red) and fair dropping (black), and the ActiveList 99^{th} percentile on the right y-axis for fair dropping as functions of load (ρ = flow arrival rate $\times$ mean flow cycles cost / CPU capacity). Throughput behavior is broadly as expected: fair dropping yields almost ideal PS throughput for both flow types while tail dropping severely degrades the performance of the flows with lower CPU cycle cost, especially at high loads.

To explain observed unfairness of FD at low load, consider the throughput of an isolated full-rate flow emitting packets at 10% above the nominal CPU rate. The drop rate d_1 for cost-1 flows would be such that $1.1(.5d_1 + (1-d_1)) = 1$,

i.e., the drop rate is such that packet arrival rate $\times$ average cost is equal to CPU capacity. This yields $d_1 = 2/11$ and a corresponding flow throughput of 0.9. A similar calculation for cost-10 flows yields a throughput of 0.99. The loss rate for single-packet flows is negligible with fair dropping but rises to around 10% with tail dropping.

Results for the ActiveList 99^{th}percentile are quite different to those of Fig. 7. This is due to batch processing and the impact of single-packet flows. All single-packet flows in a batch bring a new ActiveList flow (that will be removed on the next batch arrival). The number of such flows depends on the batch size and is added to the small number of active full-rate flows that is accurately predicted by the PS model. Fair dropping remains scalable in that the number of active flows remains very small compared to the possibly large number of flows in progress.

VI. Conclusion

Applying proposed fair dropping algorithms in a software router has been shown to realize per-flow fair sharing of both bandwidth and CPU. The algorithms are scalable because the number of flows to be managed is small (less than 200 with high probability at normal loads) whatever the link speed or CPU capacity. It is compatible with batch I/O and batch processing, optimizations which significantly impede the implementation of classical schedulers.

The algorithms have been successfully implemented in the VPP software router that is part of the FD.io Linux Foundation project. Preliminary experimental results show them to be effective and relatively lightweight in terms of induced overhead.

There is, however, a significant overhead involved in selectively dropping packets within the CPU, due mainly to packet I/O, whatever the algorithm employed. This overhead can compromise performance when flows are non-responsive to congestion. We are currently investigating extensions to our approach where non-responsive flows can be effectively dealt with before their packets are polled by the CPU.

Acknowledgments

This work has been carried out at LINCS (http://www.lincs.fr) and benefited from support of NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech (http://newnet.telecom-paristech.fr).

References

[1] https://fd.io/wp-content/uploads/sites/34/2017/07/FDioVPPwhitepaperJuly2017.pdf.

[2] http://dpdk.org/doc/guides/prog_guide/qos_framework.html.

[3] Data plane development kit. http://dpdk.org.

[4] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *USENIX NSDI*, 2012.

[5] T. Barbette, C. Soldani, and L. Mathy. Fast userspace packet processing. In *ACM/IEEE ANCS*, 2015.

[6] S. Benfredj, T. Bonald, A. Proutiere, G. Régnié, and J. W. Roberts. Statistical bandwidth sharing: A study of congestion at flow level. In *ACM SIGCOMM*, 2001.

[7] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *ACM SIGCOMM*, 1996.

[8] T. Bonald and J. Roberts. Multi-resource fairness: Objectives, algorithms and performance. In *ACM SIGMETRICS*, 2015.

[9] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. On multi–gigabit packet capturing with multi–core commodity hardware. In *Passive and Active Measurement*, pages 64–73. Springer, 2012.

[10] F. Fusco and L. Deri. High Speed Network Traffic Analysis with Commodity Multi-core Systems. In *ACM IMC*, 2010.

[11] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle. Comparison of frameworks for high-performance packet io. In *ACM/IEEE ANCS*, 2015.

[12] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. In *ACM SIGCOMM*, 2012.

[13] P. Goyal, H. Vin, and H. Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Trans. on Netw.*, 5(5):690–704, Oct 1997.

[14] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290, Jan. 2018.

[15] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC Research Report TR-301, 1984.

[16] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon. NBA (Network Balancing Act): A High-performance Packet Processing Framework for Heterogeneous Processors. In *ACM European Conference on Computer Systems (EuroSys)*, 2015.

[17] W.-J. Kim and B. G. Lee. Fred-fair random early detection algorithm for tcp over atm networks. *Electronics Letters*, 34(2):152–154, Jan 1998.

[18] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts. Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing. In *ACM SIGMETRICS*, 2005.

[19] A. Kortebi, S. Oueslati, and J. Roberts. Implicit service differentiation using deficit round robin. In *Proceedings of ITC19*, 2005.

[20] L. Linguaglossa, D. Rossi, D. Barach, D. Marjon, and P. Pfiester. High-speed software data plane via vectorized packet processing. Tech report, 2017.

[21] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 192–201, Nov 2001.

[22] J. Nagle. On packet switches with infinite storage. RFC 970, 1985.

[23] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM Comput. Commun. Rev.*, 33(2):23–39, Apr. 2003.

[24] R. Pan, B. Prabhakar, and K. Psounis. Choke - a stateless active queue management scheme for approximating fair bandwidth allocation. In *INFOCOM*, 2000.

[25] L. Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference*, pages 101–112, 2012.

[26] M. Shin, S. Chong, and I. Rhee. Dual-resource tcp/aqm for processing-constrained networks. *IEEE/ACM Trans. Netw.*, 16(2):435–449, Apr. 2008.

[27] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *ACM SIGCOMM Comput. Commun. Rev.*, 25(4):231–242, Oct. 1995.

[28] K. To, D. Firestone, G. Varghese, and J. Padhye. Measurement based fair queuing for allocating bandwidth to virtual machines. In *ACM HotMiddlebox*, 2016.

[29] L. Vasilescu, V. Olteanu, and C. Raiciu. Sharing cpus via endpoint congestion control. In *Proceedings of the Workshop on Kernel-Bypass Networks*, KBNets '17, pages 31–36, New York, NY, USA, 2017. ACM.

A New Dependence Model for Heterogeneous Markov Modulated Poisson Processes

Fang Dong, Kui Wu, Venkatesh Srinivasan
Department of Computer Science, University of Victoria, B.C., Canada

***Abstract*—Markov Modulated Poisson Process (MMPP) has been extensively studied in random process theory and widely applied in various applications involving Poisson arrivals whose rate varies following a Markov process. The most general form of aggregated MMPP is the superposition of heterogeneous MMPPs (HeMMPP), in which each constituent MMPP has different parameters. Due to the generality of HeMMPP, studying its temporal dependence will benefit network traffic monitoring and traffic prediction. Modeling the temporal dependence of HeMMPP, however, is extremely hard because the total number of states in a HeMMPP increases exponentially with the number of states in constituent MMPPs. This paper tackles the above challenge with copula analysis. It not only presents a novel framework to capture the functional dependence structure of HeMMPP, but also provides a recursive algorithm to effectively calculate HeMMPP copula values. The theoretical analysis and the algorithms together offer a complete solution for modeling the temporal dependence of HeMMPP. Another contribution of the paper is the application of HeMMPP copula for traffic prediction.**

I. Introduction

Markov modulated Poisson process (MMPP) is a doubly stochastic Poisson process whose arrival rate is modulated by an irreducible continuous time Markov chain (CTMC) independent with the arrival process [6]. MMPP was first proposed by Yechiali and Naor to model non-homogeneous Poisson arrival process in queueing systems [17]. Specifically, the arrival process is a Poisson process with arrival rate λ_j whenever the CTMC is in state j. MMPP can effectively capture burst arrivals and sudden changes in arrivals since it can integrate significantly different rates into one model. This advantage makes MMPP a widely applied model for the arrival processes of network systems [13]. When multiple independent MMPP flows, each having a different set of parameters, arrive in a system, the total arrivals become the superposition of independent heterogeneous MMPPs (HeMMPP).

From the theoretical aspect, HeMMPP is the most general form of MMPP aggregate, because single MMPP and the superposition of independent homogeneous MMPPs (HoMMPP) are both special cases of HeMMPP. Therefore, the study of HeMMPP can benefit both real-world applications and theoretical performance analysis. *From the practical aspect*, HeMMPP traffic exists in many real-world applications. For instance, HeMMPP has been applied to generate self-similar traffic to Internet backbone [19]. HeMMPP can be used to capture multiple multimedia sources to a multimedia server because each multimedia traffic can be reasonably modeled with MMPP [16].

The temporal dependence of HeMMPP can help develop fitting methods that model real network traffic trace with HeMMPP [1] or predict the trend of arrivals [19]. Despite its importance, however, the dependence structure of HeMMPP is largely unknown. It can be shown that HeMMPP is still an MMPP [6], but analysing HeMMPP as one MMPP becomes intractable due to the exponential increase in the number of states [8]. For instance, modeling a HeMMPP consisting of two 20-state MMPPs is computationally difficult [8]. In other words, simply treating the superposition of MMPPs as one MMPP with a larger number of states will not work well. As such, we need to develop a different method to analyse the temporal dependence of HeMMPP.

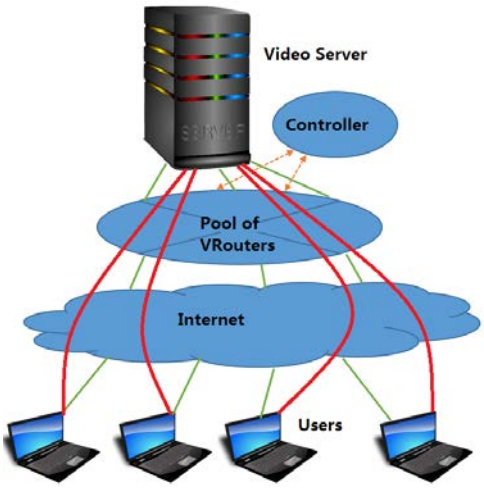


Fig. 1: Motivating example: a content delivery system relies on software-defined network and a pool of virtual network functions (e.g., VRouters) to adjust network bandwidth aligning with demand of the multimedia flows (denoted by red curves)

Astute readers may observe an alternative method to avoid the need of analysing HeMMPP: we may simply track and study each constituent MMPP, and in the context of traffic prediction, we predict the future arrivals of each constituent MMPP and aggregate them as the prediction of future HeMMPP arrivals. We call this alternative the *predict-individual* method. This method, however, poses extra burden in practice. In an example shown in Fig. 1, a content delivery system relies on software-defined networks and a pool of virtual network functions (e.g., VRouters) to adjust network

ISBN 978-3-903176-08-9

bandwidth aligning with the multimedia flows, each labelled by a red curve and modeled as an MMPP [16]. Using the predict-individual method, the controller needs to keep record of arrivals of each MMPP and makes prediction on each MMPP. In contrast, with a HeMMPP model, the controller tracks arrivals of each MMPP *only during the HeMMPP modeling process*. Once the HeMMPP model is built, the controller does not need to track individual MMPP flows and instead uses the aggregated arrivals to predict future HeMMPP arrivals. Clearly, the HeMMPP model greatly simplifies the tasks of the controller. In addition, using HeMMPP model for prediction leads to more accurate results than using the predict-individual method, as shown in our later evaluation.

In the state-of-art temporal dependence model of HeMMPP, the covariance between number of arrivals in different time slots (called arrival counts) in constituent MMPP is derived asymptotically in [1]. The summation of asymptotic covariance of constituent MMPPs turns to be the approximate covariance between arrival counts in HeMMPP. Nevertheless, there is a large gap towards obtaining a complete temporal dependence model of HeMMPP for two reasons. First, most papers consider HeMMPP where its constituent MMPPs have only two states [1], [15]. The temporal dependence of HeMMPP, whose constituent MMPPs have a higher number of states, needs further study. Second, covariance or autocovariance is only capable of measuring *linear* dependence over time, but the network traffic traces may exhibit much more complex dependence than that. We are thus motivated to search for a functional dependence structure of HeMMPP, which carries richer and more complete information of dependence.

We tackle the problem with copula, an advanced dependence measure that links marginals into joint distribution. This paper analyses the functional dependence between arrival counts in HeMMPP and makes the following contributions:

1) It uses a new dependence measure, copula, to analyse the dependence structure of HeMMPP. The copula-based dependence reveals functional temporal dependence, which is more powerful than the commonly-used measures for linear dependence, covariance and correlation.
2) The copula-based analysis can effectively deal with the difficulty in modeling HeMMPP, where each constituent MMPP may have an arbitrary number of states.
3) It not only presents a recursive algorithm to compute the theoretical copula values of HeMMPP, but also adopts parametric copulas to model the temporal dependence of HeMMPP.
4) It demonstrates an application of HeMMPP dependence model in traffic prediction.

II. Preliminaries

A. Markov Modulated Poisson Process

We introduce the definition and key concepts of MMPP and HeMMPP.

Definition 1. *A **Markov-modulated Poisson Process** (MMPP) [6] is constructed by varying the arrival rate of a Poisson process according to an m-state irreducible continuous-time Markov chain (CTMC). In particular, when the Markov Chain is in state j, the arrivals follow a Poisson process of rate λ_j. Therefore, an MMPP can be parameterized by the Q matrix [14] of CTMC and the m Poisson arrival rates, $\Lambda = (\lambda_1, \ldots, \lambda_m)$.*

We thus denote an MMPP by parameters (Q, Λ).

Definition 2. ***Environment-stationarity** of an MMPP [6]: An MMPP (Q, Λ) is considered to be environment-stationary if its associated CTMC Q is stationary.*

For an environment-stationary MMPP, the stationary distribution of the states, $\Pi = (\pi_1, \ldots, \pi_m)$, is determined by solving the equation $\Pi Q = 0$.

Definition 3. ***HeMMPP**: An MMPP is called **HeMMPP** if it is a superposition of multiple independent heterogeneous MMPPs. The constituent MMPPs carry different parameters $({}_1Q, {}_1\Lambda), ..., ({}_rQ, {}_r\Lambda)..., ({}_lQ, {}_l\Lambda)$, where $({}_rQ, {}_r\Lambda)$ denotes the parameters of the r-th constituent MMPP.*

To distinguish regular MMPP with superposition of MMPPs, we use the term *single MMPP* to refer to an MMPP not created from superposition, the term *HeMMPP* to refer to aggregate MMPPs containing multiple single MMPPs, and each single MMPP in HeMMPP is called *constituent MMPP*. In this paper, we only consider stationary MMPPs, which means each constituent MMPP in HeMMPP is environment-stationary.

For ease of reference, the main notations used in the paper are listed in Tables I, II and III. Note that we consider HeMMPP with l number of constituent MMPPs. When $l = 1$, HeMMPP degrades to single MMPP, and in this case l can be omitted from notation. Thus A_i, M, C and $C_{i'}$ are notations for single MMPP.

Remark 1. *Due to the intricate composition of HeMMPP, a complicate and slightly unconventional notation system is necessary. We, however, adopt the following rules to make the notations easy to follow: the superscript denotes the number of constituent MMPPs, the right hand-side subscript denotes the time slot-related information (or random variables from the context), and the left hand-side subscript denotes the index of a specific constituent MMPP in consideration.*

B. Copulas

A copula is a function that links univariate marginals to their multivariate distribution. The definition of 2-copula is:

Definition 4. ***(Copula)** A 2-dimensional copula is a function C having the following properties [10]:*

1) *Its domain is $[0, 1] \times [0, 1]$;*
2) *C is 2-increasing, i.e., for every $u_1, u_2, v_1, v_2 \in [0, 1]$ and $u_1 \leq u_2, v_1 \leq v_2$, we have $C(u_2, v_2) - C(u_2, v_1) - C(u_1, v_2) + C(u_1, v_1) \geq 0$.*
3) *$C(u, 0) = C(0, v) = 0$, $C(u, 1) = u$, $C(1, v) = v$, for every $u, v \in [0, 1]$.*

TABLE I: Parameters for Single MMPP

Notation	Explanation
Q	The transition rate matrix of associated CTMC
Λ	The vector of Poisson arrival rates
Π	The stationary distribution of asscoiated CTMC
$P(t)$	The transition matrix after time t of associated CTMC

TABLE II: Notations of HeMMPP

Notation	Explanation
A_i^l	The arrival count in i-th time slot of HeMMPP consisting of l number of consituent MMPPs
$M^l(x)$	The marginal distribution of A_i^l
$C^l(u,v)$	The copula between A_i^l and A_{i+1}^l
$\nabla C^l(u,v)$	The copula gradient of $C^l(u,v)$
$\mathbb{C}^l$	The copula matrix for $C^l(u,v)$
$\mathbb{R}^l$	The copula gradient matrix for $\nabla C^l(u,v)$
$C_{i'}^l(u,v)$	The copula between A_i^l and $A_{i+i'}^l$
$C_{i'}^l(u,v;\theta)$	The parametric copula between A_i^l and $A_{i+i'}^l$

Theorem 1. ***(Sklar's theorem)*** *[10] Let H be a joint distribution function with marginals F_X and F_Y, then there exists a copula C such that for for all x and y,*

$$H(x,y) = Pr(X \le x, Y \le y) = C(F_X(x), F_Y(y)).$$

Sklar's theorem is the core of the copula theory. First, it shows how the copula connects marginals with joint distribution. This property is especially useful since the joint distribution of random variables is hard to find directly in many applications [10]. In this situation, integration of a copula model and marginals makes it easy to understand the joint behaviour. Second, Sklar's theorem implies that copula, as a dependence measure, is entirely separated from both marginals and joint distribution.

The dependence in terms of copula is stable when the marginals changes functionally. This beautiful feature is formally stated in the following theorem:

Theorem 2. ***(The invariant property of copulas)*** *[10] Let X and Y be continuous random variables with copula C_{XY}. If α_1 and α_2 are strictly increasing functions on the range of X and the range of Y, respectively, then $C_{\alpha_1(X)\alpha_2(Y)} = C_{XY}$. In other words, C_{XY} is invariant under strictly increasing transformations of X and Y.*

There are mainly two methods to build a copula model. The first method is to construct a theoretical copula for the problem at hand. The inversion method belongs to this category.

Theorem 3. ***(Inversion method)*** *[10] Let H be a joint distribution function with marginals F_X and F_Y. Let F_X^{-1} and F_Y^{-1} be the inverse function of F_X and F_Y. Then the copula between X and Y can be constructed as*

$$C(u,v) = H(F_X^{-1}(u), F_Y^{-1}(v)) \quad \forall u, v,$$

such that

$$H(x,y) = C(F_X(x), F_Y(y)) \quad \forall x, y.$$

The other method to build a copula model is to fit real data into known parametric copulas. A large variety of parametric copulas are available for parametric copula modeling, for instance, Gaussian copula, Student's copula, Clayton copula, Frank copula, and Gumbel copula. Since the theoretical copula is not always easy to derive, parametric copula modeling has become popular in practice [7]. A copula model built with this method is also called an parametric copula.

TABLE III: Notations of the r-th Constituent MMPP

Notation	Explanation
$({}_rQ, {}_r\Lambda)$	The parameters of the r-th MMPP
${}_rA_i$	The arrival count in i-th time slot of r-th MMPP trace
${}_rM(x)$	The marginal distribution of ${}_rA_i$
${}_rp(x)$	The probability mass function of ${}_rA_i$
${}_rC(u,v)$	The copula between ${}_rA_i$ and ${}_rA_{i+1}$
$\nabla_r C(u,v)$	The copula gradient of ${}_rC(u,v)$
${}_rC_{i'}(u,v)$	The copula between ${}_rA_i$ and ${}_rA_{i+i'}$

Copula-based dependence is tightly associated with tail dependence measure. The tail dependence comprises upper tail dependence given by

$$\begin{aligned}\rho_t^+ &= \lim_{u\to 1} Pr(X > F_X^{-1}(u) | Y > F_Y^{-1}(u)) \\ &= \lim_{u\to 1} \frac{1-2u+C(u,u)}{1-u};\end{aligned} \tag{1}$$

and the lower tail dependence given by

$$\rho_t^- = \lim_{u\to 0} Pr(X < F^{-1}(u) | Y < F^{-1}(u)) = \lim_{u\to 0} \frac{C(u,u)}{u}. \tag{2}$$

Copula is promising for modeling temporal dependence of HeMMPP for the following reasons: copula can be constructed theoretically or by parametric copula modeling; copula is a functional dependence model and captures rounded dependence information beyond linear scope; the invariant property keeps copula structure stable when HeMMPP scales functionally. In this paper, we will study both theoretical copula (Section III) and parametric copula of HeMMPP (Section IV).

III. Theoretical Copula Analysis for HeMMPP

A. Theoretical Results for Single MMPP

A functional dependence model of single MMPP has been investigated in [4], from which some results are useful for our study of HeMMPP. To make this paper self-contained, we summarize these results below.

In the analysis of single MMPP with parameters (Q, Λ) [4], the time is divided into equal-sized small intervals, called time slots. The length of each time slot is denoted as Δ, which is short enough such that the state transition of MMPP within one time slot is negligible[1]. Denote the sequence of time slots as $I_1, I_2, \ldots, I_n$, and the number of arrivals in I_i as A_i. The sequence $\{A_i\}$ is also called arrival counts as introduced in Section I. Denote the transition matrix by $P(t) = [p_{j_1 j_2}(t)]$, where $p_{j_1 j_2}(t)$ is the probability that the CTMC switches from state j_1 to state j_2 after time t. The transition matrix $P(t)$ can be calculated with well-known methods such as those

[1]This assumption is justified since the arrival rate in one time slot is (approximately) stable when Δ is small.

introduced in Chapter 6.8 of [14]. The stationary distribution of CTMC is $\Pi = (\pi_1, \pi_2, ..., \pi_m)$. The marginal distribution function of A_i is given in Theorem 4, and the copula between A_i and A_{i+1} is given in Theorem 5. The multi-step copula between A_i and $A_{i+i'}$ is given in Theorem 6. These theoretical marginals and copulas can be applied to the constituent MMPP in HeMMPP.

Theorem 4. *[4] The marginal distribution of A_i is*

$$M(x) \equiv Pr(A_i \leq x) = \sum_{j=1}^{m} \pi_j G_j(x) \tag{3}$$

where $G_j(x) = e^{-\lambda_j \Delta} \sum_{k=0}^{k=x} \frac{(\lambda_j \Delta)^k}{k!}$

Theorem 5. ***(MMPP copula)*** *[4] The copula of A_i and A_{i+1} can be calculated as:*

$$C(u, v) = \mathbb{G}(M^{-1}(u)) diag(\Pi) P(\Delta) \mathbb{G}(M^{-1}(v))^T, \tag{4}$$

where

- $\mathbb{G}(x) \equiv [G_1(x), \cdots, G_m(x)]$ *is a vector,*
- M^{-1} *is the inverse function of* M *defined by (3),*
- $diag(\Pi)$ *is a square diagonal matrix with the elements of vector* Π *on the main diagonal,*
- $\mathbb{G}(M^{-1}(v))^T$ *is the transpose of* $\mathbb{G}(M^{-1}(v))$.

Theorem 6. ***(Multi-step MMPP copula)*** *[4] The copula of A_i and $A_{i+i'}$ in a single MMPP can be calculated as:*

$$C_{i'}(u, v) = \mathbb{G}(M^{-1}(u)) diag(\Pi) P(i'\Delta) \mathbb{G}(M^{-1}(v))^T. \tag{5}$$

B. Theoretical Copula for HeMMPP

1) Theoretical Analysis for HeMMPP Copula: Since the constituent MMPPs in HeMMPP possess different parameters, we have to differentiate the constituent MMPPs by numbering them. We randomly select an order of constituent MMPPs, i.e., $({}_1Q, {}_1\Lambda), ({}_2Q, {}_2\Lambda), ..., ({}_rQ, {}_r\Lambda)..., ({}_lQ, {}_l\Lambda)$, where $({}_rQ, {}_r\Lambda)$ represents the parameters of the r-th constituent MMPP ($r = 1, 2, \cdots, l$). For constituent MMPPs, ${}_rA_i$, ${}_rM$, ${}_rp$, ${}_rC$, $\nabla_r C$ denote arrival counts, marginal CDF, marginal probability mass function (PMF), copula and copula gradient of r-th MMPP, respectively. In HeMMPP, A_i^l, C^l, M^l are notations of the superposition of the *first* l number of constituent MMPPs. Note that we introduce this ordering for ease of explanation, and the order will not influence our analytical results. The following theorems show how to analyse theoretical marginal and copula of HeMMPP.

Theorem 7. *The HeMMPP marginal distribution function has recursive relationship between M^l and $M^{l-1} (l \geq 2)$ as*

$$M^l(x) = \sum_{x'=0}^{x} M^{l-1}(x - x') * {}_lp(x'), \tag{6}$$

where ${}_lp$ is the probability mass function (PMF) of the arrival count from l-th MMPP, ${}_lp(x') = {}_lM(x') - {}_lM(x' - 1)$.

Proof. The key idea of the proof is to divide the arrivals from l number of MMPPs into the arrivals from the first $l - 1$ number of MMPPs plus the arrivals from the l-th MMPP, i.e., $A_i^l = A_i^{l-1} + {}_lA_i$. Thus, we have

$$\begin{aligned} M^l(x) = Pr(A_i^l \leq x) &= \sum_{x'=0}^{x} Pr(A_i^l \leq x | {}_lA_i = x') Pr({}_lA_i = x') \\ &= \sum_{x'=0}^{x} Pr(A_i^{l-1} \leq x - x') Pr({}_lA_i = x') \\ &= \sum_{x'=0}^{x} M^{l-1}(x - x') * {}_lp(x') \end{aligned}$$

□

Theorem 8. *The HeMMPP copula has the recursive relationship between C^l and C^{l-1} as shown below:*

$$\begin{aligned} C^l(M^l(x), M^l(y)) = &\sum_{x'=0}^{x} \sum_{y'=0}^{y} \nabla_l C({}_lM(x'), {}_lM(y')) \\ &* C^{l-1}(M^{l-1}(x - x'), M^{l-1}(y - y')), \end{aligned} \tag{7}$$

where $\nabla_l C$ is the copula gradient of the l-th MMPP. The copula gradient of r-th MMPP ($r = 1, 2, \cdots, l$) is defined as

$$\begin{aligned} &\nabla_r C({}_rM(x'), {}_rM(y')) \\ \equiv &{}_rC({}_rM(x'), {}_rM(y')) + {}_rC({}_rM(x' - 1), {}_rM(y' - 1)) \\ &- {}_rC({}_rM(x'), {}_rM(y' - 1)) - {}_rC({}_rM(x' - 1), {}_rM(y')). \end{aligned} \tag{8}$$

Proof. The proof is also on the basis of $A_i^l = A_i^{l-1} + {}_lA_i$.

$$\begin{aligned} &C^l(M^l(x), M^l(y)) \\ =&Pr(A_i^l \leq x, A_{i+1}^l \leq y) \\ =&\sum_{x'=0}^{x} \sum_{y'=0}^{y} Pr(A_i^l \leq x, A_{i+1}^l \leq y | {}_lA_i = x', {}_lA_{i+1} = y') \\ &* Pr({}_lA_i = x', {}_lA_{i+1} = y') \\ =&\sum_{x'=0}^{x} \sum_{y'=0}^{y} Pr(A_i^{l-1} \leq x - x', A_{i+1}^{l-1} \leq y - y') \\ &* Pr({}_lA_i = x', {}_lA_{i+1} = y') \\ =&\sum_{x'=0}^{x} \sum_{y'=0}^{y} C^{l-1}(M^{l-1}(x - x'), M^{l-1}(y - y')) \\ &* Pr({}_lA_i = x', {}_lA_{i+1} = y') \end{aligned}$$

Since the arrival counts follow discrete distribution and the domain is non-negative integers, for all non-negative integer x' and y', and $\forall r \in \{1, 2, \cdots, l\}$, we have

$$\begin{aligned} &Pr({}_rA_i = x', {}_rA_{i+1} = y') \\ =&Pr({}_rA_i \leq x', {}_rA_{i+1} \leq y') + Pr({}_rA_i \leq x' - 1, {}_rA_{i+1} \leq y' - 1) \\ &- Pr({}_rA_i \leq x', {}_rA_{i+1} \leq y' - 1) - Pr({}_rA_i \leq x' - 1, {}_rA_{i+1} \leq y') \\ =&{}_rC({}_rM(x'), {}_rM(y')) + {}_rC({}_rM(x' - 1), {}_rM(y' - 1)) \\ &- {}_rC({}_rM(x'), {}_rM(y' - 1)) - {}_rC({}_rM(x' - 1), {}_rM(y')) \\ =&\nabla_r C({}_rM(x'), {}_rM(y')). \end{aligned}$$

Therefore, by replacing $Pr({}_lA_i = x', {}_lA_{i+1} = y')$ with $\nabla_l C({}_lM(x'), {}_lM(y'))$, we can calculate the copula C^l as:

$$C^l(M^l(x), M^l(y)) = \sum_{x'=0}^{x}\sum_{y'=0}^{y} \nabla_l C({}_lM(x'), {}_lM(y')) * C^{l-1}(M^{l-1}(x-x'), M^{l-1}(y-y')).$$

□

Theorems 7 and 8 reveal the relationship between M^l and M^{l-1} and relationship between C^l and C^{l-1}. Even with these relationships, it is still hard to derive the closed-form HeMMPP copula. However, they are sufficient for developing recursive algorithms to numerically calculate HeMMPP copula, as introduced in the next section.

2) Recursive Algorithms for Calculating HeMMPP Copula: To design algorithms to calculate HeMMPP copula efficiently, we narrow down the interesting range of A_i^l from its infinite domain to finite range with an upper threshold $\hat{a}$. In other words, although the range of A_i^l is on the whole non-negative integer domain, we only need to compute the copula values $C^l(M^l(x), M^l(y))$ for $x < \hat{a}$ and $y < \hat{a}$. The selection of $\hat{a}$ is application dependent and can be set appropriately based on the trace data. Narrowing down the interesting range makes the computation feasible and still meets practical needs, because the arrival counts in real traffic flows always fall within a limited range.

On the interesting range $[0, \hat{a})$, we define seven matrices in Table IV. $\mathbb{M}^l$, $\mathbb{C}^l$ and $\mathbb{R}^l$ are for HeMMPP, and they represent HeMMPP marginal values, HeMMPP copula values and HeMMPP copula gradient values, respectively. For instance, the number in row x of matrix $\mathbb{M}^l$ represents the value of $M^l(x-1)$, *i.e.*, $\mathbb{M}^l_x \equiv M^l(x-1)$. For the constituent MMPPs, their values in PMF, marginal values, copula values and copula gradient values are represented by ${}_r\mathbb{P}$, ${}_r\mathbb{M}$, ${}_r\mathbb{C}$ and ${}_r\mathbb{R}$, respectively, as shown in Table IV. To emphasize the matrices' dimension, we mark dimensions on the bottom right, such as $[\mathbb{M}^l]_{\hat{a}}$ and $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$. We also use a notation $[\mathbb{C}^l]_{x\times y}$ to represent the submatrix of $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$ with its first x rows and first y columns.

With HeMMPP parameters $({}_1Q, {}_1\Lambda), ({}_2Q, {}_2\Lambda), ..., ({}_lQ, {}_l\Lambda)$ and a properly-set threshold value $\hat{a}$, we design Algorithm 1 (with the time complexity as $O(\hat{a} \times l)$) to calculate HeMMPP marginal matrix $[\mathbb{M}^l]_{\hat{a}}$ and Algorithm 2 (with the time complexity as $O(\hat{a} \times \hat{a} \times l)$) to calculate HeMMPP copula matrix $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$. The recursive procedure in Algorithm 1, **MARG, implements Theorem 7;** the recursive procedure in Algorithm 2, **CPA, implements Theorem 8**. With matrices $[\mathbb{M}^l]_{\hat{a}}$ and $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$ computed, the theoretical copula of HeMMPP is revealed in Theorem 9:

Theorem 9. ***(HeMMPP copula)*** *Given HeMMPP with marginal matrix $[\mathbb{M}^l]_{\hat{a}}$ and copula matrix $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$, its copula value of $C^l(u, v)$ can calculated by*

$$C^l(u, v) = \mathbb{C}^l_{(\text{argmax}_x\, \mathbb{M}^l_x \le u)(\text{argmax}_y\, \mathbb{M}^l_y \le v)} \quad (9)$$

for any u and v satisfying $u \le \mathbb{M}^l_{\hat{a}}$, $v \le \mathbb{M}^l_{\hat{a}}$.

Algorithm 1 An algorithm to compute marginal matrix $\mathbb{M}^l$

Require: the upper threshold $\hat{a}$, HeMMPP parameters $({}_1Q, {}_1\Lambda), ({}_2Q, {}_2\Lambda), ..., ({}_lQ, {}_l\Lambda)$,
Ensure: $[\mathbb{M}^l]_{\hat{a}}$

return MARG($[{}_1Q, ..., {}_lQ]$, $[{}_1\Lambda, ..., {}_l\Lambda]$, $\hat{a}$)

procedure MARG($[{}_1Q, ..., {}_lQ]$, $[{}_1\Lambda, ..., {}_l\Lambda]$, $\hat{a}$)
  $l \leftarrow$ the vector length of $[{}_1Q, ..., {}_lQ]$ or of $[{}_1\Lambda, ..., {}_l\Lambda]$
  // Base Case
  if $l == 1$ **then**
    $[\mathbb{M}^1]_{\hat{a}} \leftarrow$ compute with parameters ${}_1\Lambda$ and ${}_1Q$ based on Theorem 4
    return $[\mathbb{M}^1]_{\hat{a}}$
  end if
  // Inductive Step
  $[\mathbb{M}^{l-1}]_{\hat{a}} \leftarrow$ MARG($[{}_1Q, ..., {}_{l-1}Q]$, $[{}_1\Lambda, ..., {}_{l-1}\Lambda]$, $\hat{a}$)
  $[{}_l\mathbb{M}]_{\hat{a}} \leftarrow$ compute with parameters ${}_l\Lambda$ and ${}_lQ$ based on Theorem 4
  $[{}_l\mathbb{P}]_{\hat{a}} \leftarrow$ compute from $[{}_l\mathbb{M}]_{\hat{a}}$ based on its definition
  for $x \leftarrow 1, \hat{a}$ **do**
    Rotate matrix $[{}_l\mathbb{P}]_x$ 180 degree clockwise as $[{}_l\mathbb{P}']_x$
    Calculate Hadamard product of $[\mathbb{M}^{l-1}]_x$ and $[{}_l\mathbb{P}']_x$ as $[\mathbb{T}]_x$
    $\mathbb{M}^l_x \leftarrow$ sum of all elements in matrix $[\mathbb{T}]_x$
  end for
  return $[\mathbb{M}^l]_{\hat{a}}$
end procedure

C. Multi-step Theoretical Copulas for HeMMPP

Theorem 10. ***(Multi-step HeMMPP copula)****. The copula of A_i^l and $A_{i+i'}^l$ in HeMMPP can be constructed by integrating the multi-step MMPP copula in Theorem 6 into the recursive method in Theorem 8. Specifically, all copulas C^l in the recursive method are replaced by multi-step copulas $C^l_{i'}$, and all constituent copulas ${}_rC$ are replaced by ${}_rC_{i'}$.*

IV. Parametric Copula modeling for HeMMPP

In this section, we construct parametric copulas for HeMMPP. The parametric copulas we investigated include the following three Archimedean copulas [10]:

1) Clayton copula ($\theta \in [-1, \infty) \setminus \{0\}$)
$$C(u, v; \theta) = [\max\{u^{-\theta} + v^{-\theta} - 1, 0\}]^{-1/\theta},$$
2) Frank copula ($\theta \in [-\infty, \infty) \setminus \{0\}$)
$$C(u, v; \theta) = -\tfrac{1}{\theta}\log[1 + \tfrac{(\exp(-\theta u)-1)(\exp(-\theta v)-1)}{\exp(-\theta)-1}],$$
3) Gumbel copula ($\theta \in [1, \infty)$)
$$C(u, v; \theta) = \exp[-((-\log u)^\theta + (-\log v)^\theta)^{1/\theta}].$$

We investigate the above parametric copulas due to two reasons. First, they are all one-parameter copulas which make modelling easy. Second, they capture different types of tail dependence efficiently. The tail dependence features of the three copulas are distinct with each other: Clayton copula

TABLE IV: Definition of Matrices in HeMMPP

Matrix Denotation	Matrix name	Number in row x (and column y)
$[\mathbb{M}^l]_{\hat{a}}$	marginal matrix of HeMMPP	$\mathbb{M}^l_x \equiv M^l(x-1)$
$[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$	copula matrix of HeMMPP	$\mathbb{C}^l_{xy} \equiv C^l(M^l(x-1), M^l(y-1))$
$[\mathbb{R}^l]_{\hat{a}\times\hat{a}}$	copula gradient matrix of HeMMPP	$\mathbb{R}^l_{xy} \equiv \nabla C^l(M^l(x-1), M^l(y-1))$
$[{}_r\mathbb{M}]_{\hat{a}}$	marginal matrix of r-th MMPP	${}_r\mathbb{M}_x \equiv {}_rM(x-1) = Pr({}_rA_i \leq x-1)$
$[{}_r\mathbb{P}]_{\hat{a}}$	PMF matrix of r-th MMPP	${}_r\mathbb{P}_x \equiv {}_rp(x-1) = {}_r\mathbb{M}_x - {}_r\mathbb{M}_{x-1}$
$[{}_r\mathbb{C}]_{\hat{a}\times\hat{a}}$	copula matrix of r-th MMPP	${}_r\mathbb{C}_{xy} \equiv {}_rC({}_rM(x-1), {}_rM(y-1))$
$[{}_r\mathbb{R}]_{\hat{a}\times\hat{a}}$	copula gradient matrix of r-th MMPP	${}_r\mathbb{R}_{xy} \equiv \nabla {}_rC({}_rM(x-1), {}_rM(y-1))$

Algorithm 2 An algorithm to compute copula matrix $\mathbb{C}^l$

Require: the upper threshold $\hat{a}$, HeMMPP parameters $({}_1Q, {}_1\Lambda), ({}_2Q, {}_2\Lambda), ..., ({}_lQ, {}_l\Lambda)$
Ensure: $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$

return CPA($[{}_1Q, ..., {}_lQ], [{}_1\Lambda, ..., {}_l\Lambda], \hat{a}$)

procedure CPA($[{}_1Q, ..., {}_lQ], [{}_1\Lambda, ..., {}_l\Lambda], \hat{a}$)
  $l \leftarrow$ the vector length of $[{}_1Q, ..., {}_lQ]$ or of $[{}_1\Lambda, ..., {}_l\Lambda]$
  // Base Case
  if $l == 1$ **then**
    $[\mathbb{C}^1]_{\hat{a}\times\hat{a}} \leftarrow$ compute with parameters ${}_1\Lambda$ and ${}_1Q$ based on Theorem 5
    return $[\mathbb{C}^1]_{\hat{a}\times\hat{a}}$
  end if
  // Inductive Step
  $[\mathbb{C}^{l-1}]_{\hat{a}\times\hat{a}} \leftarrow$ CPA($[{}_1Q, ..., {}_{l-1}Q], [{}_1\Lambda, ..., {}_{l-1}\Lambda], \hat{a}$)
  $[{}_l\mathbb{C}]_{\hat{a}\times\hat{a}} \leftarrow$ compute with parameters ${}_l\Lambda$ and ${}_lQ$ based on Theorem 5
  $[{}_l\mathbb{R}]_{\hat{a}\times\hat{a}} \leftarrow$ compute from $[{}_l\mathbb{C}]_{\hat{a}\times\hat{a}}$ based on its definition
  for $x \leftarrow 1, \hat{a}$ **do**
    for $y \leftarrow 1, \hat{a}$ **do**
      Rotate matrix $[{}_l\mathbb{R}]_{x\times y}$ 180 degree clockwise to be $[{}_l\mathbb{R}']_{x\times y}$
      Calculate Hadamard product of $[\mathbb{C}^{l-1}]_{x\times y}$ and $[{}_l\mathbb{R}']_{x\times y}$ as $[\mathbb{T}]_{x\times y}$
      $\mathbb{C}^l_{xy} \leftarrow$ sum of all elements in matrix $[\mathbb{T}]_{x\times y}$
    end for
  end for
  return $[\mathbb{C}^l]_{\hat{a}\times\hat{a}}$
end procedure

models lower tail dependence; Gumbel copula models upper tail dependence; and Frank copula captures symmetric upper and lower tail dependence. Therefore, these three copulas are investigated as simple alternatives of theoretical copulas. To further improve copula fitting of HeMMPP, a mixture of these copulas or some other types of parametric copulas might be needed for modelling, which is, however, beyond the scope of this paper and a possible topic for extended research.

We follow three main steps to model the parametric copula:

1) Compute the tail dependence by definitions in Eq.(1) and Eq. (2).
2) Choose the parametric copula for HeMMPP modeling based on tail dependence:
   a) choose Clayton copula if $\rho_t^+ \approx 0$ and $\rho_t^- > 0$;
   b) choose Frank copula if $\rho_t^+ \approx \rho_t^-$;
   c) choose Gumbel copula if $\rho_t^+ > 0$ and $\rho_t^- \approx 0$.
3) Determine the value of the parameter θ for the chosen parametric copula. The parameter θ is learnt by fitting the HeMMPP trace into the chosen copula with the maximum likelihood estimation method. Specifically, θ of parametric HeMMPP copula $C^l_{i'}(u, v; \theta)$ is determined by fitting the sample pairs of (A^l_i, $A^l_{i+i'}$) of trace.

V. Application: Traffic Prediction Based on HeMMPP Copula

So far, we have shown how the full temporal dependence structure of HeMMPP can be captured with copulas. A deep understanding of the temporal dependence structure of HeMMPP can benefit many applications, e.g., dynamic resource provisioning, and self-similar traffic modeling. Another obvious application is to predict future traffic based on the temporal dependence in arrivals. This section illustrates a method to achieve this goal.

The problem of traffic prediction could be in different forms. In this paper, we focus on estimating the future arrival count $A^l_{i+i'}$ based on the current observation of arrival count A^l_i. The prediction is made by maximizing the conditional probability $Pr(A^l_{i+i'}|A^l_i)$. When $i' = 1$, the prediction is made one-step forward; when $i' > 1$, the prediction is made multi-step forward. In this section, we introduce the prediction methods with both theoretical and parametric HeMMPP copula.

Prediction based on theoretical HeMMPP copula is made according to Theorem 11:

Theorem 11. *(1) Consider a HeMMPP having theoretical copula C^l between A^l_i and A^l_{i+1}. If $A^l_i = x$ is the current observation from the arrival process and if the prediction is made by maximizing the conditional probability $Pr(A^l_{i+1}|A^l_i)$, the predicted arrival count $\hat{A}^l_{i+1}$ is:*

$$\hat{A}^l_{i+1} = \underset{y}{\operatorname{argmax}} \nabla C^l(M^l(x), M^l(y)). \tag{10}$$

(2) Consider a HeMMPP having theoretical copula $C^l_{i'}$ between A^l_i and $A^l_{i+i'}$. If $A^l_i = x$ is the current observation from the arrival process and if the prediction is made by maximizing the conditional probability $Pr(A^l_{i+i'}|A^l_i)$, the predicted arrival count $\hat{A}^l_{i+i'}$ is:

$$\hat{A}^l_{i+i'} = \operatorname*{argmax}_y \nabla C^l_{i'}(M^l(x), M^l(y)). \tag{11}$$

∇C^l and $\nabla C^l_{i'}$ are copula gradients defined in the same way as in Eq.(8) by replacing ${}_rC$ with C^l or $C^l_{i'}$ and ${}_rM$ with M^l.

Proof. We only prove part (1), because part (2) can be proved in the same way. Since the prediction is made by maximizing the conditional probability $Pr(A^l_{i+1}|A^l_i)$, we have

$$\begin{aligned}\hat{A}_{i+1} &= \operatorname*{argmax}_y Pr(A^l_{i+1} = y | A^l_i = x) \\ &= \operatorname*{argmax}_y \frac{Pr(A^l_i = x, A^l_{i+1} = y)}{Pr(A^l_i = x)} \\ &= \operatorname*{argmax}_y \frac{\nabla C^l(M^l(x), M^l(y))}{Pr(A^l_i = x)} \\ &= \operatorname*{argmax}_y \nabla C^l(M^l(x), M^l(y))\end{aligned}$$

□

Prediction based on parametric HeMMPP copula is made in the similar way of theoretical HeMMPP copula, as shown in Theorem 12. The proof is omitted because the proof idea is the same as that of Theorem 11.

Theorem 12. *Consider a HeMMPP having parametric copula $C^l_{i'}(u, v; \theta)$ between A^l_i and $A^l_{i+i'}$. If $A^l_i = x$ is the current observation from the arrival process and if the prediction is made by maximizing the conditional probability $Pr(A^l_{i+i'}|A^l_i)$, the predicted arrival count $\hat{A}^l_{i+i'}$ is:*

$$\hat{A}^l_{i+i'} = \operatorname*{argmax}_y \nabla C^l_{i'}(M^l(x), M^l(y); \theta), \tag{12}$$

where $\nabla C^l_{i'}(M^l(x), M^l(y); \theta)$ is copula gradient defined in the same way as in Eq. (8) by using copula $C^l_{i'}(M^l(x), M^l(y); \theta)$ and marginal M^l.

VI. Experimental Evaluation

A. Evaluation Methods

We have proposed both theoretical copula modeling and parametric copula modeling for traffic prediction in Section V. To evaluate the new model, we implement another prediction model, linear predictive coding (LPC(1)), for comparison. LPC(1) will be constructed from trace data. With LPC(1), the multi-step prediction from A^l_i is made as

$$\hat{A}^l_{i+1} = \gamma A^l_i, \quad \hat{A}^l_{i+2} = \gamma \hat{A}^l_{i+1}, \quad \cdots, \quad \hat{A}^l_{i+i'} = \gamma \hat{A}^l_{i+i'-1},$$

where γ is the parameter of LPC(1) model.

LPC(1) model predicts data based on the dependence information in terms of autocorrelation. Thus it is set as the benchmark predictor to show *how functional dependence modeling with copulas improves over linear dependence.* Note that the first order of LPC model is used here for a fair comparison: our copula-based prediction model is first order in the sense that only dependence between two arrival counts is considered each step. It has been shown that copula models outperform AR(1) model in MMPP traffic prediction [4]. Due to space limit, however, we omit this comparison.

We also implement and compare the *predict-individual* method, where we predict the future arrivals of each constituent MMPP separately and aggregate them as the prediction of future HeMMPP arrivals.

When applying any of the prediction models on a traffic trace, the trace is divided into two parts, the training set and the testing set. The training set comes from the first certain percentage data of the trace, and the rest of the trace constitutes the testing set. The prediction accuracy is measured by root-mean-square error (RMSE) across the testing set:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{A}^l_i - A^l_i)^2}, \tag{13}$$

where A^l_i is arrival counts from testing set at timeslot i, $\hat{A}_i$ denotes the corresponding predicted value, and n is the total number of time slots in the testing period. For a prediction model, its performance is measured by its average RMSE (aRMSE) on a trace with different training percentages. The performance improvement ratio (IMP RATIO) over benchmark model (LPC(1)) are defined as Eq.(14).

$$\text{IMP RATIO} = \frac{\text{aRMSE}_{\text{benchmark}} - \text{aRMSE}}{\text{aRMSE}_{\text{benchmark}}} * 100\%. \tag{14}$$

B. Evaluation Results

We evaluate the benefit of using HeMMPP dependence model for traffic prediction with synthetic data. We generate a HeMMPP trace using two MMPP models, each obtained by fitting the model to real-world trace. For this purpose, we follow the work [4] and use the same Bellcore traces[2], that record millions of packet arrivals on an Ethernet at Bellcore Morristown Research and Engineering facility. The traces are well known in network traffic modeling, and many papers have shown that Bellcore traces are well characterized by MMPP [1], [9]. We choose two of these traces to determine reasonable parameters for simulation of synthetic HeMMPP trace. With the fitting algorithm in [8], BCpAug89 trace is well fitted into a 12 state MMPP [4] with parameters (Q_A, Λ_A) as shown in Eq. (15); BCpOct89 trace is fitted into a 13 state MMPP with parameters (Q_O, Λ_O) listed in Eq. (16). We generate synthetic HeMMPP data consisting of two MMPP traces, by simulating each MMPP trace using the learnt parameters and aggregating the two MMPP traces.

The length of time slots is set as $\Delta = 1$ second. A^l_i denotes the number arrival of the aggregate trace in i-th second. We will study one-step dependence between A^l_i and A^l_{i+1} and two-step dependence between A^l_i and A^l_{i+2}, and conduct one-step prediction and two-step prediction accordingly.

[2]available from the website http:// ita.ee.lbl.gov/html/contrib/BC.html

$$Q_A = \begin{pmatrix} -0.857 & 0.286 & 0.429 & 0.143 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.067 & -0.900 & 0.267 & 0.233 & 0.233 & 0.067 & 0.033 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.023 & 0.078 & -0.836 & 0.336 & 0.203 & 0.102 & 0.078 & 0.000 & 0.016 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.026 & 0.140 & -0.720 & 0.274 & 0.153 & 0.085 & 0.029 & 0.007 & 0.007 & 0.000 & 0.000 \\ 0.002 & 0.008 & 0.051 & 0.173 & -0.650 & 0.244 & 0.122 & 0.041 & 0.006 & 0.002 & 0.002 & 0.000 \\ 0.000 & 0.001 & 0.027 & 0.073 & 0.173 & -0.696 & 0.303 & 0.094 & 0.014 & 0.009 & 0.001 & 0.000 \\ 0.000 & 0.001 & 0.004 & 0.019 & 0.099 & 0.233 & -0.616 & 0.200 & 0.048 & 0.012 & 0.001 & 0.000 \\ 0.000 & 0.000 & 0.008 & 0.023 & 0.049 & 0.184 & 0.409 & -0.775 & 0.084 & 0.015 & 0.003 & 0.000 \\ 0.000 & 0.000 & 0.008 & 0.015 & 0.015 & 0.120 & 0.301 & 0.218 & -0.805 & 0.113 & 0.015 & 0.000 \\ 0.000 & 0.020 & 0.000 & 0.000 & 0.059 & 0.059 & 0.235 & 0.078 & 0.275 & -0.824 & 0.098 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.077 & 0.231 & 0.231 & 0.154 & 0.077 & -0.846 & 0.077 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & -1.000 \end{pmatrix}, \tag{15}$$

$$\Lambda_A = \big(782.069, 674.207, 574.345, 482.483, 398.621, 322.759, 254.897, 195.035, 143.173, 99.311, 63.449, 35.587\big).$$

$$Q_O = \begin{pmatrix} -1.00 & 0.75 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.04 & -0.64 & 0.26 & 0.25 & 0.06 & 0.02 & 0.00 & 0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.13 & -0.72 & 0.34 & 0.16 & 0.03 & 0.03 & 0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.01 & 0.06 & 0.12 & -0.68 & 0.31 & 0.13 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.10 & 0.25 & -0.74 & 0.20 & 0.11 & 0.06 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.04 & 0.09 & 0.23 & -0.71 & 0.20 & 0.10 & 0.03 & 0.02 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.03 & 0.06 & 0.31 & -0.68 & 0.16 & 0.08 & 0.02 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.02 & 0.04 & 0.19 & 0.34 & -0.81 & 0.16 & 0.05 & 0.01 & 0.01 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.01 & 0.04 & 0.09 & 0.23 & 0.29 & -0.83 & 0.14 & 0.04 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 0.02 & 0.07 & 0.22 & 0.28 & -0.80 & 0.13 & 0.05 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.13 & 0.21 & 0.33 & -0.71 & 0.04 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.17 & 0.50 & 0.17 & -0.83 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & -1.00 \end{pmatrix}, \tag{16}$$

$$\Lambda_O = \big(1125.89, 995.67, 873.46, 759.24, 653.02, 554.81, 464.59, 382.37, 308.15, 241.94, 183.72, 133.50, 91.28\big).$$

1) One-step Prediction on HeMMPP trace: Theoretical HeMMPP copula, parametric HeMMPP copula and LPC(1) model are constructed for one-step prediction. Theoretical HeMMPP copula is computed with Algorithms 1 and 2. Based on the observations of the HeMMPP trace, the threshold for marginal and copula matrix computation is chosen as $\hat{a} = 1500$. The probability that the arrival count A_i^l exceeds the threshold is less than 0.01, *i.e.*, $Pr(A_i^l > \hat{a}) < 0.01$, indicating that there are very few observations appearing beyond the chosen threshold. Fig. 2 shows the contour of the-oretical one-step HeMMPP copula calculated with parameters $(Q_A, \Lambda_A, Q_O, \Lambda_O)$.

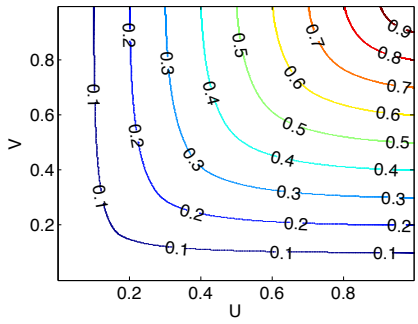


Fig. 2: The contour of theoretical one-step HeMMPP copula

Parametric HeMMPP copula is constructed through the modeling steps illustrated in Section IV. As the upper tail dependence is close to lower tail dependence ($\rho_t^+ = 0.3212$, $\rho_t^- = 0.2228$), Frank copula is chosen and its parameter is determined by fitting the training set of trace. Similarly, the parameter of LPC(1) model is determined according to the training set of the HeMMPP trace.

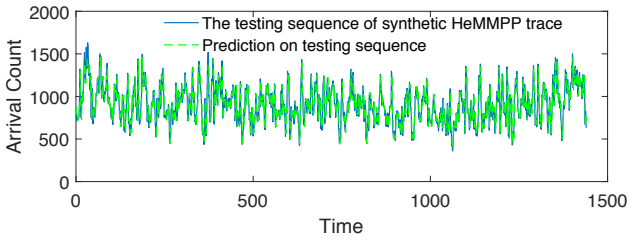


Fig. 3: One-step prediction with theoretical HeMMPP copula.

TABLE V: One-Step Prediction RMSE on the HeMMPP trace with Different Training Percentage.

Training Percentage	Theoretical Copula	Parametric Copula	Predict Individual	LPC(1)
50%	129.8456	129.9517	133.2521	190.6223
60%	129.1777	127.7076	132.0675	190.2156
70%	130.7527	129.9172	134.2631	193.6528
80%	129.4203	128.5920	133.5243	191.6844
90%	124.9783	125.2756	128.6330	188.7742
aRMSE	128.8349	128.2888	132.3480	190.9899
IMP RATIO	32.54%	32.83%	30.70%	—

With different percentage of trace data for training, three models are constructed accordingly and applied for one-step prediction. Fig. 3 shows the prediction with theoretical HeMMPP copula on the test set of the last 20% arrival counts. The detailed prediction errors are shown in terms of RMSE in Table V. The smaller value of RMSE represents the more accurate prediction. aRMSE shows the average performance over different training percentage (from 50% to 90%). IMP RATIO shows that **copula-based predictions, including theoretical**

copula model and parametric copula model, have more than 30% improvement over the LPC(1) model, showing the advantage of functional dependence modeling (such as copulas) over linear dependence measurement (such as autocorrelation). In addition, we can see that HeMMPP-based prediction works better than the predict-individual method. This is because any prediction includes errors and the predict-individual method may aggregate the errors from predictions in individual MMPP. This phenomenon is more obvious in the two-step prediction results shown in Table VI.

2) Two-step Prediction on HeMMPP trace: Two-step prediction is also performed to evaluate multi-step dependence modeling. Similar to the previous section, theoretical copula, parametric copula and LPC(1) models are constructed for two-step prediction. Table VI compares the two-step prediction performance of copulas with LPC(1) model. **Our copula models have a great improvement ratio (nearly 30%) over LPC(1) model regarding the two-step predictions.**

TABLE VI: Two-Step Prediction RMSE on the HeMMPP trace with Different Training Percentage.

Training Percentage	Theoretical Copula	Parametric Copula	Predict Individual	LPC(1)
50%	174.8214	175.6217	195.0492	246.4333
60%	174.4992	174.4648	192.8411	245.1160
70%	176.9906	176.0216	197.3482	252.5563
80%	174.9567	174.0762	194.7852	247.4094
90%	170.2201	169.0609	194.4664	246.7953
aRMSE	174.2976	173.8490	194.8980	247.6621
IMP RATIO	29.62%	29.80%	21.30%	—

VII. Related Work

The research related to our work applies covariance to model the temporal dependence among single MMPP or HeMMPP. The covariance between arrival counts in single MMPP is derived in [11]. The closed-form covariance between arrival counts in two-state MMPP is given in [1]. The work closest to ours is [4], where temporal dependence in single MMPP and in superposition of homogeneous MMPPs has been investigated. Nevertheless, HeMMPP is more complicated and more general, and its temporal dependence has not been studied in [4].

In the case of HeMMPP, the constituent MMPPs can be combined into one MMPP with formula given in [6]. Even though some efforts have made to reduce the number of states to approximate HeMMPP [8], [18], very few works consider HeMMPP as one MMPP for calculation of covariance. In [1], asymptotic covariance of two-state MMPPs is summed to get an approximate covariance of the superposition of MMPPs. Our work is different from the above work as we build the functional temporal dependence of arrival counts in HeMMPP with copula.

Copula models have been broadly used in the domain of financial analysis, for multivariate dependence modeling [3] as well as for time series modeling [12]. It has attracted attention in the domain of networks in recent years [2], [4], [5].

VIII. Conclusion

With copula analysis, this paper is the first that theoretically derives the intricate temporal dependence structure in HeMMPP. It not only presents a complete solution for modeling functional dependence in HeMMPP, but also introduces parametric copulas as fast approximation of theoretical copulas. Using the theoretical and parametric copulas, we show the value of our research in an example application, i.e., traffic prediction. While the study of MMPP has a long history and the topic of MMPP might not be trending, the novel theoretical results and the new algorithms developed in this paper will benefit a broad class of current and future network applications involving MMPP traffic flows.

References

[1] A. T. Andersen and B. F. Nielsen. A markovian approach for modeling packet traffic with long-range dependence. *IEEE Journal on Selected Areas in Communications*, 16(5):719–732, 1998.

[2] K. Avrachenkov, N. M. Markovich, and J. K. Sreedharan. Distribution and dependence of extremes in network sampling processes. *Computational Social Networks*, 2(1):1, 2015.

[3] M. Beil. *Modeling dependencies among financial asset returns using copulas*. PhD thesis, Technische Universität München, 2013.

[4] F. Dong, K. Wu, and V. Srinivasan. Copula analysis of temporal dependence structure in markov modulated poisson process and its applications. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2(4), 2017.

[5] F. Dong, K. Wu, and S. Venkatesh. Copula analysis for statistical network calculus. In *Proceedings of INFOCOM*, pages 1535–1543, Hong Kong, 2015. IEEE.

[6] W. Fischer and K. Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Performance Evaluation*, 18(2):149–171, 1993.

[7] C. Genest, B. Rémillard, and D. Beaudoin. Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics*, 44(2):199–213, 2009.

[8] D. P. Heyman and D. Lucantoni. Modeling multiple ip traffic streams with rate limits. *IEEE/ACM Transactions on Networking*, 11(6):948–958, 2003.

[9] L. Muscariello, M. Mellia, M. Meo, M. A. Marsan, and R. L. Cigno. Markov models of internet traffic and a new hierarchical mmpp model. *Computer Communications*, 28(16):1835–1851, 2005.

[10] R. B. Nelsen. *An introduction to copulas*. Springer, New York, 2006.

[11] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Taylor & Francis, New York, USA, 1989.

[12] A. Patton. Copula methods for forecasting multivariate time series. *Handbook of economic forecasting*, 2:899–960, 2012.

[13] A. Rajabi and J. W. Wong. Provisioning of computing resources for web applications under time-varying traffic. In *2014 IEEE 22nd International Symposium on MASCOTS*, pages 152–157, Paris, France, 2014. IEEE.

[14] S. M. Ross. *Introduction to probability models*. Academic Press, Burlington, 2003.

[15] P. Salvador, R. Valadas, and A. Pacheco. Multiscale fitting procedure using markov modulated poisson processes. *Telecommunication Systems*, 23(1-2):123–148, 2003.

[16] S. Shah-Heydari and T. Le-Ngoc. Mmpp models for multimedia traffic. *Telecommunication Systems*, 15(3-4):273–293, 2000.

[17] U. Yechiali and P. Naor. Queuing problems with heterogeneous arrivals and service. *Operations Research*, 19(3):722–734, 1971.

[18] M. Yu and M. Zhou. A model reduction method for traffic described by mmpp with unknown rate limit. *IEEE Communications Letters*, 10(4):302–304, 2006.

[19] M. Yuksel, B. Sikdar, K. Vastola, and B. Szymanski. Workload generation for ns simulations of wide area networks and the internet. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 93–98, 2000.

Improving Output Bounds in the Stochastic Network Calculus Using Lyapunov's Inequality

Paul Nikolaus
Distributed Computer Systems (DISCO) Lab
TU Kaiserslautern, Germany
nikolaus@cs.uni-kl.de

Jens Schmitt
Distributed Computer Systems (DISCO) Lab
TU Kaiserslautern, Germany
jschmitt@cs.uni-kl.de

***Abstract*—Giving tight estimates for output bounds is key to an accurate network analysis using the stochastic network calculus (SNC) framework. In order to upper bound the delay for a flow of interest in the network, one typically has to calculate output bounds of cross-traffic flows several times. Thus, an improvement in the output bound calculation pays off considerably. In this paper, we propose a new output bound calculation in the SNC framework by making use of Lyapunov's inequality. We prove the bound's validity and also show why it is always at least as accurate as the state-of-the-art method. Numerical evaluations demonstrate that even in small heterogeneous two server topologies, our approach can improve a delay bounds' violation probability by a factor of 340. For a set of randomly generated parameters, the bound is still decreased by a factor of 1.33 on average. Furthermore, our approach can be easily integrated in existing end-to-end analyses.**

I. Introduction

A. Motivation

Providing delay bounds in packet-switched networks is a timeless challenge with recent sample applications as, e.g., Internet at the speed of light [1], Tactile Internet [2], Internet of Things [3], or the envisioned cyber-physical systems [4], which often face real-time requirements.

The Network Calculus (NC) holds the promise to enable tight end-to-end delay analysis in such advanced applications building on a modular and uniform mathematical framework based on min-plus algebra [5]. Starting from the 1990s with two papers by Cruz [6], [7], NC demonstrated its benefits providing tight bounds for deterministic worst-case end-to-end delay bounds. In the following, the Deterministic Network Calculus (DNC) was further elaborated and mathematically cast into a min-plus algebra setting [8], [9]. More recently, NC was generalized into a stochastic setting providing probabilistic worst-case bounds: the Stochastic Network Calculus (SNC) framework [8], [10]–[13]. SNC's main features can be summarized as providing a very general scheduling abstraction (the service curve) and the ability to enable system-wide end-to-end analysis (the concatenation theorem) [13].

SNC results can be categorized into different branches such as tail-bound based [10], [12], [14], moment generating functions (MGF) based [8], [11], and martingale based [15] approaches. Recent work evidences its applicability to modern problems, e.g., in the analysis of parallel systems (using the fork-join pattern) or multi-tenancy [16]–[18].

ISBN 978-3-903176-08-9

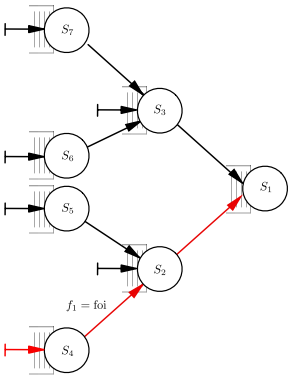


Fig. 1. Full binary sink tree with seven nodes.

Typically a DNC/SNC network analysis proceeds along the following steps:

1) Reducing the network to a tandem of servers traversed by the flow of interest (foi) by invoking the output bound calculation to characterize cross-traffic flows at the servers where they join the foi.
2) Reducing the tandem of servers traversed by the flow of interest (foi) to a single server representing the whole system.
3) Calculating the delay bound of the foi at the single server representing the whole system.

Most of the existing NC literature has mainly focused on steps 2) and 3). In DNC, step 1) has seen some advanced treatment recently [19], but in SNC it has been largely neglected in the sense that no work beyond the standard output bound calculation was invested. In contrast to this, we focus on step 1) and, in particular, try to improve the SNC output bound calculation in this paper. As the output bound calculation has to be invoked numerous times in step 1), we believe its accuracy to be key in larger network analyses. For example: Assume a full binary tree of height h where each node represents a server and each of these servers has an arrival flow that is transmitted to the sink; let the foi be starting from one of the leaf nodes (see also Figure 1), then the number of output bound calculations is $2^h - h - 1$, whereas we only

need to invoke the delay bound calculation once (in step 3)). Thus, any improvement in the output bound calculation pays off tremendously in larger network analyses.

Yet, how can we improve upon the standard SNC output bound calculation? The tail bound and MGF SNC analyses have the application of the so-called Union bound or Boole's inequality in common. In a series of publications, [15], [20]–[22], the authors emphasized its poor performance and suggested an appealing martingale-based approach. It provides tight single hop lower and upper bounds on the delay for different scheduling disciplines. Yet, to the best of our knowledge, so far there is no concatenation result in the martingale-based SNC and thus step 2) from above cannot be performed and, thus, an elegant end-to-end analysis remains elusive. Hence, we decided to remain within the standard SNC framework and, yet, try to counteract the inherent problems of the Union bound.

B. Main Contribution

In this paper, we present a modification of the MGF-based SNC that mitigates the Union bound's effect in the output bound calculation. It consists of the application of Lyapunov's inequality just before the invocation of the Union bound and does not impose any additional assumptions. It is thus minimally invasive and all existing results and procedures of the SNC are literally still applicable while, as we see below, it improves the performance bounds. In fact, we prove this new bound to be always at least as good as the state-of-the-art method and show that in a very simple heterogeneous two server setting, it can improve the delay bound already by a factor of up to 340.

It comes, however, at the price of an additional parameter per invocation of Lyapunov's inequality. Thus, we trade higher computational effort in the optimization of these parameters for improved bounds. However, as we also show this effort is moderate if the optimization is done carefully.

C. Outline

The rest of the paper is structured as follows: In Section II, we introduce the necessary notations for SNC and its main results as we need them in this paper. In Section III, we present our new output bound calculation and prove its validity. A numerical evaluation is given in Section IV: we compare output bounds for a single server and delay bounds for a two server setting as well as a fat tree topology with the current state of the art method. In Section V, we prove that Lyapunov's inequality cannot be applied directly to delay bounds. Section VI concludes the paper.

II. SNC Background and Notation

In this section, we introduce some of the basic terms and concepts in SNC.

We use the MGF-based SNC in order to calculate per-flow delay bounds. To be precise, we bound the probability that the delay exceeds a given value, typically denoted by T. The connection between probability bounds and MGFs is established by Chernoff's bound

$$\mathrm{P}(X > a) \leq e^{-\theta a}\,\mathrm{E}\left[e^{\theta X}\right], \quad \theta > 0, \tag{1}$$

with $\mathrm{E}\left[e^{\theta X}\right]$ as the moment-generating function (MGF) of a random variable X. We define an *arrival flow* by the stochastic process A with discrete time space $\mathbb{N}$ and continuous state space $\mathbb{R}_0^+$ as $A(s,t) \coloneqq \sum_{i=s+1}^{t} a\,(i)$, with $a(i)$ as the traffic increment process in time slot i. Network calculus provides an elegant system-theoretic analysis by employing min-plus algebra:

Definition 1 (Convolution in Min-Plus Algebra)**.** The min-plus (de-)convolution of real-valued, bivariate functions $x(s,t)$ and $y(s,t)$ is defined as

$$\begin{aligned}(x \otimes y)\ (s,t) &\coloneqq \min_{s \leq i \leq t} \left\{x\,(s,i) + y(i,t)\right\}, \\ (x \oslash y)\ (s,t) &\coloneqq \max_{0 \leq i \leq s} \left\{x(i,t) - y(i,s)\right\}.\end{aligned} \tag{2}$$

The characteristics of the service process are captured by the notion of a dynamic S-server.

Definition 2 (Dynamic S-Server)**.** Assume a service element has an arrival flow A as its input and the respective output is denoted by A'. Let $S(s,t)$, $0 \leq s \leq t$, be a stochastic process that is nonnegative and increasing in t. The service element is a *dynamic S-server* iff for all $t \geq 0$ it holds that:

$$A'(0,t) \geq (A \otimes S)\ (0,t) = \min_{0 \leq i \leq t} \left\{A(0,i) + S(i,t)\right\}.$$

The analysis in this paper is based on a per-flow perspective. I.e., we consider a certain flow, the so-called *flow of interest* (foi). Throughout this paper, for the sake of simplicity, we assume the servers' scheduling to be arbitrary multiplexing [23]. That is, if flow f_2 is prioritized over flow f_1, the leftover service for the corresponding arrival A_1 is $S_{\text{l.o.}} = [A_2 - S]^+$. Furthermore, we require the server to be work-conserving.

In the following definition, we introduce (σ,ρ)-constraints [8] as they enable us to give stationary bounds under stability conditions.

Definition 3 ((σ,ρ)-Bound)**.** An arrival flow is $(\sigma_A(\theta), \rho_A(\theta))$-bounded for some $\theta > 0$, if its MGF exists and for all $0 \leq s \leq t$

$$\mathrm{E}\left[e^{\theta A(s,t)}\right] \leq e^{\theta(\rho_A(\theta)(t-s)+\sigma_A(\theta))}.$$

A dynamic S-server is $(\sigma_S(\theta), \rho_S(\theta))$-bounded for some $\theta > 0$, if its MGF exists and for all $0 \leq s \leq t$

$$\mathrm{E}\left[e^{-\theta S(s,t)}\right] \leq e^{\theta(\rho_S(\theta)(t-s)+\sigma_S(\theta))}.$$

Definition 4 (Virtual Delay)**.** The *virtual delay* at time $t \geq 0$ is defined as

$$d(t) \coloneqq \inf\left\{s \geq 0 : A(0,t) \leq A'(0,t+s)\right\}.$$

It can briefly be described as the time it takes for the cumulated departures to "catch up with" the cumulated arrivals.

Theorem 5 (Output and Delay Bound)**.** *[11] [24] Consider an arrival process $A(s,t)$ with dynamic S-server $S(s,t)$.*

The departure process A' is upper bounded for any $0 \leq s \leq t$ according to

$$A'(s,t) \leq (A \oslash S)\ (s,t).$$

The delay at $t \geq 0$ is upper bounded by

$$d(t) \leq \inf\left\{s \geq 0 : (A \oslash S)\ (t+s,t) \leq 0\right\}.$$

We focus on the analogue of Theorem 5 for moment generating functions:

Theorem 6 (Output and Delay MGF-Bound)**.** *[11] [24] For the assumptions as in Theorem 5, we obtain:*

The MGF of the departure process A' is upper bounded for any $0 \leq s \leq t$ according to

$$\mathrm{E}\left[e^{\theta A'(s,t)}\right] \leq \mathrm{E}\left[e^{\theta (A \oslash S)\,(s,t)}\right]. \tag{3}$$

The violation probability of a given stochastic delay bound T at time t is bounded by

$$\mathrm{P}(d(t) > T) \leq \mathrm{E}\left[e^{\theta (A \oslash S)\,(t+T,t)}\right]. \tag{4}$$

III. New Output Bound Calculation

In this section, we derive our new approach to compute the MGF-output bound. Furthermore, we apply this idea to $(\sigma(\theta), \rho(\theta))$-bounded arrivals and service.

A. Insertion of Lyapunov's Inequality

The most intuitive way to bound (3) is to continue with

$$\begin{aligned}\mathrm{E}\left[e^{\theta (A \oslash S)\,(s,t)}\right] &\overset{(2)}{=} \mathrm{E}\left[e^{\theta \max_{0\leq i\leq s}\{A(i,t)-S(i,s)\}}\right] \\ &\leq \sum_{i=0}^{s} \mathrm{E}\left[e^{\theta (A(i,t)-S(i,s))}\right], \end{aligned} \tag{5}$$

where the max is always less than or equal to the sum since we have only non-negative terms. Inequality (5) is similar to the application of the Union bound[1],

$$\mathrm{P}\left(\max_{i=1,\ldots,n} X_i > a\right) \leq \sum_{i=1}^{n} \mathrm{P}(X_i > a). \tag{6}$$

It has been shown to often perform poorly, in particular for correlated increments. The authors of [25] suggested instead a martingale-based approach that allows for significantly more accurate delay bounds. To the best of our knowledge, however, achieving a concatenation property to enable an end-to-end analysis remains an elusive goal in the martingale-based approach.

[1]For probability bounds such as the backlog or the delay, it is even equivalent to the Union bound, as

$$\begin{aligned}&\mathrm{P}\left(\max_{i=1,\ldots,n} X_i > a\right) \overset{(6)}{\leq} \sum_{i=1}^{n} \mathrm{P}(X_i > a) \overset{(1)}{\leq} e^{-\theta a} \sum_{i=1}^{n} \mathrm{E}\left[e^{\theta X_i}\right] \\ \Leftrightarrow\ &\mathrm{P}\left(\max_{i=1,\ldots,n} X_i > a\right) \overset{(1)}{\leq} e^{-\theta a}\, \mathrm{E}\left[\max_{i=1,\ldots,n} e^{\theta X_i}\right] \overset{(5)}{\leq} e^{-\theta a} \sum_{i=1}^{n} \mathrm{E}\left[e^{\theta X_i}\right]\end{aligned}$$

Therefore, we call inequality (5) in the following "quasi-Union bound."

In this paper, we use Lyapunov's inequality to mitigate the Union bound's effect. Yet, as we see in Subsection III-B, existing end-to-end analyses are still applicable.

Proposition 7 (Lyapunov Inequality)**.** *Let $X \geq 0$ be in $\mathcal{L}^l$ with $l \geq 1$. Then it holds that*

$$\mathrm{E}[X] \leq \left(\mathrm{E}\left[X^l\right]\right)^{\frac{1}{l}}. \tag{7}$$

Remark 8. Proposition 7 is a special case of *Jensen's inequality* [26]:

$$h(\mathrm{E}[X]) \leq \mathrm{E}[h(X)], \tag{8}$$

where h is a differentiable convex function on $\mathbb{R}$. The fact that X must be in $\mathcal{L}^l$ has a negligible effect since $l = 1$ should always be feasible, i.e., $\mathrm{E}[X]$ exists. As the random variables of our interest have existing MGF bounds, this should be a very mild assumption.

Since $l = 1$ is feasible for $X \in \mathcal{L}^1$, (7) can be rewritten as

$$\mathrm{E}[X] = \inf_{l\geq 1}\left\{\left(\mathrm{E}\left[X^l\right]\right)^{\frac{1}{l}}\right\}. \tag{9}$$

Using (9) one step before the quasi-Union bound's invocation (5) leads to

$$\begin{aligned}\mathrm{E}\left[e^{\theta A'(s,t)}\right] &\leq \mathrm{E}\left[e^{\max_{0\leq i\leq s}\{A(i,t)-S(i,s)\}}\right] \\ &\overset{(9)}{=} \inf_{l\geq 1}\left\{\left(\mathrm{E}\left[e^{l\theta \max_{0\leq i\leq s}\{A(i,t)-S(i,s)\}}\right]\right)^{\frac{1}{l}}\right\} \\ &\overset{(5)}{\leq} \inf_{l\geq 1}\left\{\left(\sum_{i=0}^{s} \mathrm{E}\left[e^{l\theta (A(i,t)-S(i,s))}\right]\right)^{\frac{1}{l}}\right\}. \end{aligned} \tag{10}$$

This new bound is obviously always at least as accurate as the quasi-Union bound (5), since $l = 1$ is feasible. The reason why this can improve over previous estimation lies in the subadditivity of the root function. It yields the following relation:

$$\begin{aligned}&\inf_{l\geq 1}\left\{\left(\sum_{i=0}^{s} \mathrm{E}\left[e^{l\theta (A(i,t)-S(i,s))}\right]\right)^{\frac{1}{l}}\right\} \\ \leq &\inf_{l\geq 1}\left\{\sum_{i=0}^{s}\left(\mathrm{E}\left[e^{l\theta (A(i,t)-S(i,s))}\right]\right)^{\frac{1}{l}}\right\}.\end{aligned}$$

The infimum on the right hand side is achieved at $l = 1$, which proves again that Lyapunov's inequality cannot worsen the bound's tightness. Yet, the subadditivity also implies that the insertion of Lyapunov's inequality can mitigate the effect of the quasi-Union bound (5), since we take the root outside of the sum. As our numerical evaluation in Section IV shows, in some cases a significant improvement for the output bound is achieved despite this method's minimal invasiveness.

B. Application to (σ, ρ)-Bounds

The bounds in (5) and (10) give an estimate for the min-plus operators in Theorem 6, but are computationally infeasible for larger networks. Since the number of sums in these calculations typically scales linearly with the number of invoked min-plus operators, one usually seeks for stationary closed-form

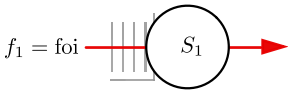


Fig. 2. One server topology.

solutions. Using (σ, ρ)-bounds (Definition 3) conveniently solves this problem by letting these sums converge, as the next proposition together with its corresponding remark show.

Proposition 9. *[24] Consider a $(\sigma_A(\theta), \rho_A(\theta))$-bounded arrival process $A(s,t)$ with $(\sigma_S(\theta), \rho_S(\theta))$-bounded dynamic S-server $S(s,t)$. If the stability condition $\rho_A(\theta) < -\rho_S(\theta)$ holds, then the output A' is $(\sigma_{A'}(\theta), \rho_{A'}(\theta))$-bounded with*

$$\sigma_{A'}(\theta) = \sigma_A(\theta) + \sigma_S(\theta) - \frac{1}{\theta}\log\left(1 - e^{\theta(\rho_A(\theta)+\rho_S(\theta))}\right),$$
$$\rho_{A'}(\theta) = \rho_A(\theta).$$

Remark 10. The computational advantage can be observed as follows:
The quasi-Union bound yields $\mathrm{E}\left[e^{\theta A'(s,t)}\right] \overset{(5)}{\leq} \sum_{i=0}^{s} \mathrm{E}\left[e^{\theta(A(i,t)-S(i,s))}\right]$, i.e., we have to compute a sum with $s+1$ summands. With the additional assumption of (σ, ρ)-constraints, the output can be bounded by the closed form $\frac{e^{\theta(\rho_A(\theta)(t-s)+\sigma_A(\theta)+\sigma_S(\theta))}}{1-e^{\theta(\rho_A(\theta)+\rho_S(\theta))}}$ (see Subsection (III-C) for details).

By an analogous calculation, we obtain for our new output bound the following result:

Proposition 11. *Under the same assumptions as in Proposition 9, under the stability condition $\rho_A(l\theta) < -\rho_S(l\theta)$ we obtain that the output A' is $(\sigma_{A'}(\theta), \rho_{A'}(\theta))$-bounded with*

$$\sigma_{A'}(\theta) = \sigma_A(l\theta) + \sigma_S(l\theta) - \frac{1}{l\theta}\log\left(1 - e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))}\right),$$
$$\rho_{A'}(\theta) = \rho_A(l\theta),$$

where $l \geq 1$.

Proof: See Appendix A. ∎

Thus, this new output bound can be also used within (σ, ρ)-constraints. I.e., it can easily be integrated in existing end-to-end analyses.

C. Single Server Example

Assume a single flow - single server setting as in Figure 2. We have already deduced that

$$\mathrm{E}\left[e^{\theta\left(A'(s,t)\right)}\right] \overset{(3)}{\leq} \mathrm{E}\left[e^{\theta(A \oslash S)(s,t)}\right] \overset{(5)}{\leq} \sum_{i=0}^{s} \mathrm{E}\left[e^{\theta(A(i,t)-S(i,s))}\right].$$

Given that the arrivals and the service have (σ, ρ)-constraints, for $\rho_A(\theta) < -\rho_S(\theta)$ we continue with

$$\mathrm{E}\left[e^{\theta\left(A'(s,t)\right)}\right] \leq \sum_{i=0}^{s} \mathrm{E}\left[e^{\theta(A(i,t)-S(i,s))}\right]$$

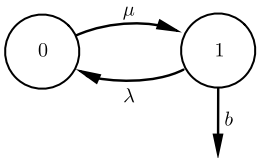


Fig. 3. MMOO Model.

$$= \sum_{i=0}^{s} \mathrm{E}\left[e^{\theta A(i,t)}\right] \mathrm{E}\left[e^{-\theta S(i,s)}\right]$$
$$\leq \sum_{i=0}^{s} e^{\theta\rho_A(\theta)(t-i)+\theta\sigma_A(\theta)} e^{\theta\rho_S(\theta)(s-i)+\theta\sigma_S(\theta)}$$
$$= e^{\theta(\rho_A(\theta)(t-s)+\sigma_A(\theta)+\sigma_S(\theta))} \cdot \sum_{j=0}^{s} e^{\theta(\rho_A(\theta)+\rho_S(\theta))j}$$
$$\leq \frac{e^{\theta(\rho_A(\theta)(t-s)+\sigma_A(\theta)+\sigma_S(\theta))}}{1 - e^{\theta(\rho_A(\theta)+\rho_S(\theta))}}, \tag{11}$$

where we have used the independence of arrivals and service in the second line, (σ, ρ)-bounds in the third line and the convergence of the geometric series in the last line.

If we used Lyapunov inequality instead, we would obtain in comparison

$$\mathrm{E}\left[e^{\theta A'(s,t)}\right] \leq \inf_{l \geq 1}\left\{\left(\frac{e^{l\theta(\rho_A(l\theta)(t-s)+\sigma_A(l\theta)+\sigma_S(l\theta))}}{1 - e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))}}\right)^{\frac{1}{l}}\right\} \leq \inf_{l \geq 1}\left\{\frac{e^{\theta(\rho_A(l\theta)(t-s)+\sigma_A(l\theta)+\sigma_S(l\theta))}}{\left(1 - e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))}\right)^{\frac{1}{l}}}\right\}. \tag{12}$$

IV. Evaluation

In this section, we investigate the increased accuracy of our new output bound introduced in Section III. That is, we evaluate the gain of the output bound calculation in a single server setting in conjunction with the delay bound for a two server topology and a fat tree. The improvement factor is measured by calculating

$$\frac{\text{Bound standard approach}}{\text{Bound new method}},$$

where clearly larger values are desirable.

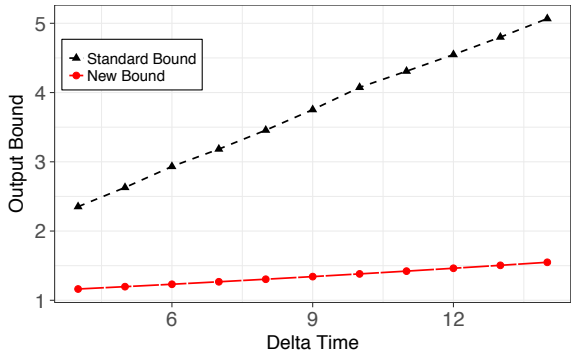
The formulae are implemented in the general-purpose programming language **Java**[2], version 8.

The arrivals are either exponentially distributed with parameter λ, i.e.,

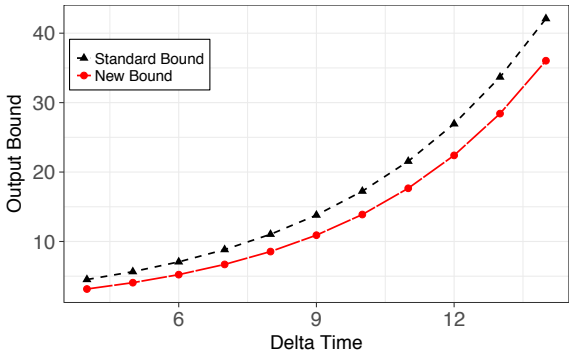
$$\mathrm{E}\left[e^{\theta A(s,t)}\right] = \left(\frac{\lambda}{\lambda - \theta}\right)^{t-s}, \quad 0 < \theta < \lambda,$$

or follow the Markov-Modulated On-Off (MMOO) traffic model. That is, it consists of a continuous-time Markov chain with two states, 0 and 1, together with transition rates μ and

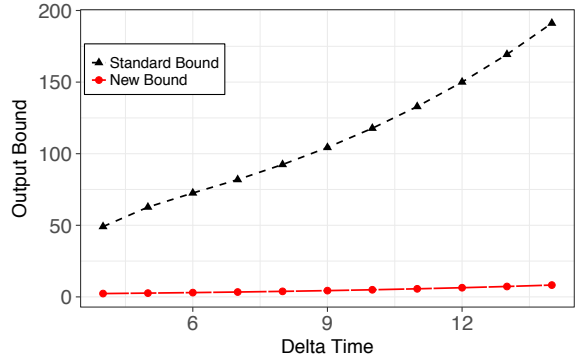
[2]https://java.com



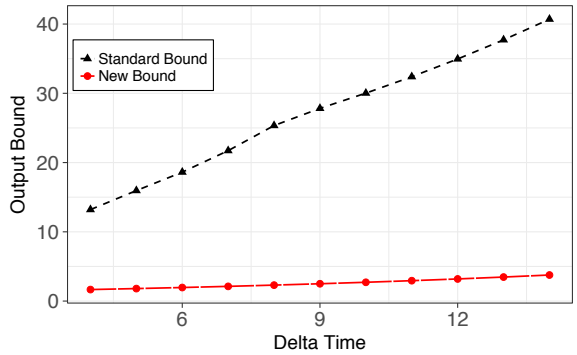
(a) Exponential arrivals with $\lambda = 3.8$, service rate $r = 3$



(b) Exponential arrivals with $\lambda = 0.5$, service rate $r = 10$



(c) MMOO with $\mu = 8$, $\lambda = 12$, $b = 3$, service rate $r = 1.5$



(d) MMOO with $\mu = 4$, $\lambda = 12$, $b = 3$, service rate $r = 1.5$

Fig. 4. Output bound comparison in the single server setting.

λ. If it is in state 0, it means that no traffic arrives, whereas in state 1, data with burst rate b are sent (see Figure 3). It has been shown in [27] that, for this arrival model, the MGF can be bounded by

$$\mathrm{E}\left[e^{\theta A(s,t)}\right] \leq e^{\theta\omega(\theta)\cdot(t-s)}, \quad \theta > 0,$$

where $\omega(\theta) = \frac{-d+\sqrt{d^2+4\mu\theta b}}{2\theta}$ and $d = \mu + \lambda - \theta b$. The service is always chosen to be work-conserving and of constant rate.

If not stated otherwise, θ and the Lyapunov parameters l_i are optimized by a naive grid search, i.e, we define points along a grid for each parameter, calculate the bound for each combination, and take the one with the best objective value.

With each application of this new inequality, an additional parameter has to be optimized. On the other hand, since the costs of incorporating Lyapunov's inequality in a given implementation are rather moderate, it gives us convenient new options: Either we prioritize accuracy and optimize all l_i (at the cost of higher computational effort), or focus more on speed setting many $l_i = 1$ (setting all l_i equal to 1 would yield the old approach). Hence, we gain more flexibility while being minimally invasive at the same time.

A. Single Server

For the single hop topology (Figure 2), we calculated the bounds in (11) and (12). For exponentially distributed arrivals and Markov-Modulated On-Off (MMOO) traffic, two examples for each distribution are depicted in Figure 4. As we can observe from these examples, the actual gain from our new output bound calculation can vary strongly depending on the scenarios' parameters. For that reason, we decided to systematically sample the parameter spaces in a Monte Carlo-type fashion. That is, we took samples from a uniform distribution as well as an exponential distribution (since the parameter space is only lower bounded) and computed the

Distribution	Average gain	Maximum gain
Exponential	1.30	1025.0
MMOO	1.34	381.9

Distribution	Average gain	Maximum gain
Exponential	1.23	233.7
MMOO	1.62	3449.9

TABLE I
OUTPUT BOUND IMPROVEMENT FOR A SINGLE SERVER (ABOVE: UNIFORM SAMPLING, BELOW: EXPONENTIAL SAMPLING).

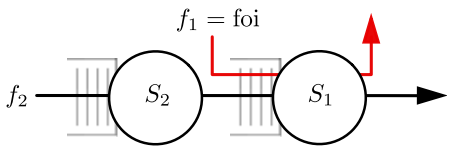
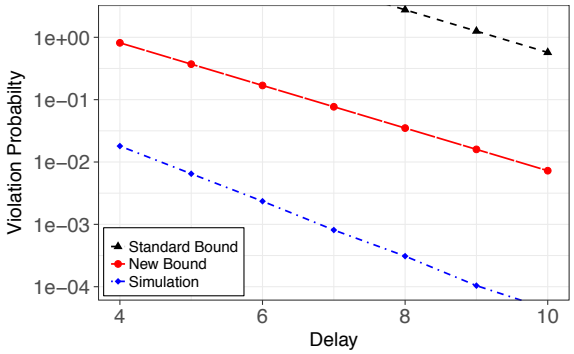
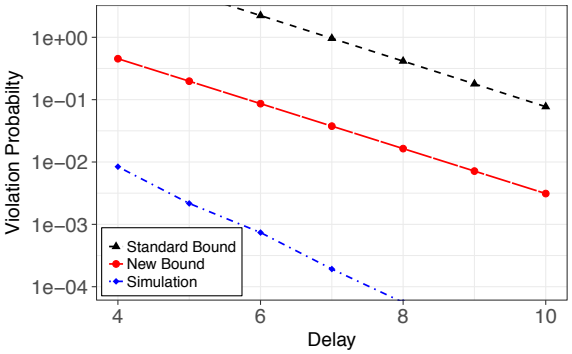


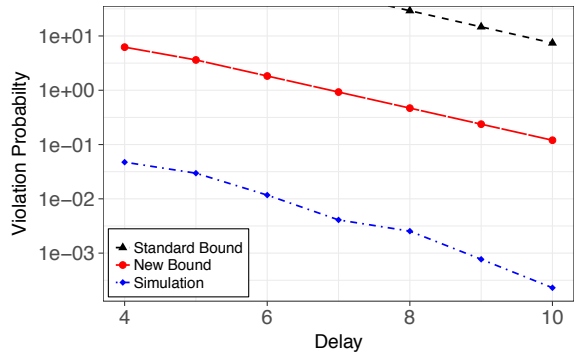
Fig. 5. Two server topology.



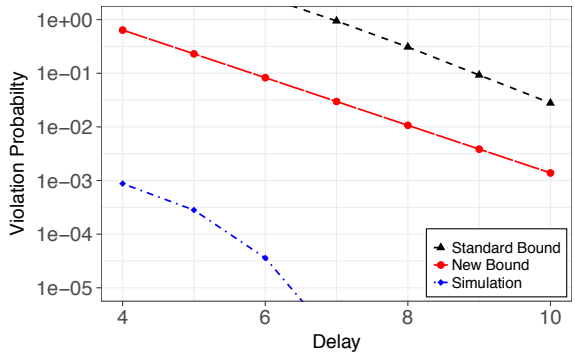
(a) Exponential arrivals with $(\lambda_1, \lambda_2) = (0.2, 8.0)$, service rates $(r_1, r_2) = (8.0, 0.2)$



(b) Exponential arrivals with $(\lambda_1, \lambda_2) = (0.4, 3.5)$, service rates $(r_1, r_2) = (4.5, 0.4)$



(c) MMOO with $(\mu_1, \mu_2) = (1.2, 3.7)$, $(\lambda_1, \lambda_2) = (2.1, 1.5)$, $(b_1, b_2) = (3.5, 0.4)$, service rates $(r_1, r_2) = (2.0, 0.3)$



(d) MMOO with $(\mu_1, \mu_2) = (1.0, 3.6)$, $(\lambda_1, \lambda_2) = (2.2, 1.6)$, $(b_1, b_2) = (3.4, 0.4)$, service rates $(r_1, r_2) = (2.0, 0.3)$

Fig. 6. Delay bound comparison in the two server setting.

average and largest improvement. The results are given in Table I.

We observe the possible gain to vary strongly with a maximum ratio Standard Bound / New Bound of three orders of magnitude. The overall average improvement factor is about 1.37, where exponentially distributed samples lead to larger improvements than the uniform ones.

B. Two Server Topology

In the previous subsection, we show that vast improvement on the output bound is possible in some cases. Next, we investigate the effect on the delay bound. Therefore, we extend the previous setting by an additional server (Figure 5). Here, a cross flow f_2 enters server S_2 and its output $(\leq (A_2 \oslash S_2))$ is prioritized over the flow of interest f_1 at server S_1. The improved output bound impacts the delay by being more accurate in terms of the foi's leftover service. Mathematically speaking, this leftover service at S_1 is described by $S_{1,\text{l.o.}} = [S_1 - (A_2 \oslash S_2)]^+$. In this topology, we calculate the delay bound (4) but take the new output bound invocation into account. Again, we display exponentially distributed arrivals and MMOO traffic. The plot is complemented by delay measurements in a packet-level simulation. Here, the violation probability is estimated by the empirical distribution computing the average number of occurred delays. All parameters are again randomly sampled by the Monte-Carlo type approach from the previous subsection.

Distribution	Average gain	Maximum gain
Exponential	1.14	255.2
MMOO	1.23	100.7

Distribution	Average gain	Maximum gain
Exponential	1.76	85.5
MMOO	1.81	342.0

TABLE II
IMPROVEMENT OF THE DELAY'S VIOLATION PROBABILITY FOR THE TWO SERVER SETTING (ABOVE: UNIFORM SAMPLING, BELOW: EXPONENTIAL SAMPLING).

As for the output bound, we often observe an improved delay bound, as one can see in the examples of Figure 6. It shows that even in the delay space (the difference in the delay bound for a given probability), the difference is up to 50%. Depending on the parameters, the gap between the simulation results and the analytically derived bounds can be closed considerably. Average behavior on the other hand is less significant. Table II indicates a highly non-linear behavior where some violation probabilities are improved by a factor of 342.0, whereas average gain is moderate with a total mean of 1.33.

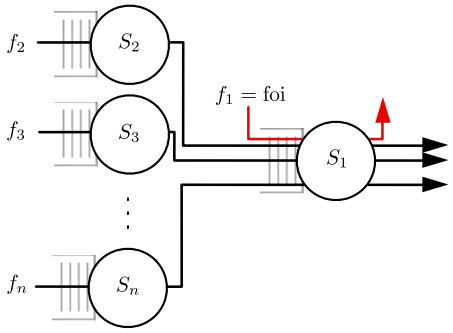


Fig. 7. Fat tree topology.

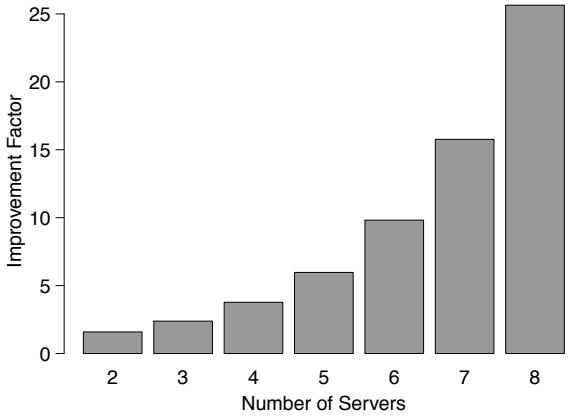


Fig. 8. Delay bound improvement for different numbers of servers.

C. Fat Tree

Starting off with the two server topology in Figure 5, we investigate the delay bound's scaling behavior for multiple invocations of Lyapunov's inequality. We now take a look at n flows, where $n-1$ are cross flows with corresponding server and their outputs jointly enter server S_1 (see Figure 7). The flow of interest is again, due to arbitrary multiplexing, assumed to be served after the cross traffic. In terms of leftover service provided for the foi, this means $S_{1,\text{l.o.}} = [S_1 - \sum_{i=2}^{n} (A_i \oslash S_i)]^+$.

We calculated the delay's violation probability for the following setting: The foi is exponentially distributed with parameter $\lambda_1 = 0.5$ and enters server S_1 with rate 4.5. The $n-1$ cross flows are also exponentially distributed, but with parameters $\lambda_i = 8$, $i = 2, \ldots, n$ and corresponding servers S_i with rates $r_i = 2$, $i = 2, \ldots, n$. The accuracy gain for different numbers of servers is depicted in Figure 8.

We observe that the ratio increases quickly to 25.6 in the case of 8 servers, even though only an improvement of 1.59 was achieved for the two server setting. This shows that the Lyapunov approach can fully develop its strengths in larger networks, when more output bound calculations have to be invoked.

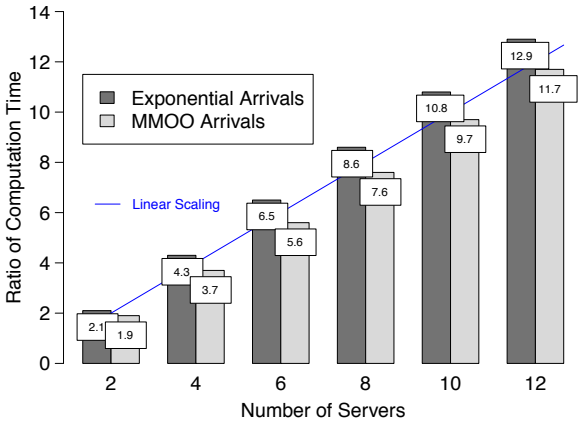


Fig. 9. Computation time comparison for the state-of-the-art and Lyapunov approach.

D. Runtime

So far, we focused on the Lyapunov bound's accuracy gain and observed favorable outcomes. Yet, the other side of the coin is the computational effort the new output bound calculation must invest to optimize over the higher-dimensional parameter space. To investigate this in more detail, we ran 10^4 experiments for exponentially distributed arrivals as well as MMOO-traffic in the two server topology (Figure 5) and the fat tree (Figure 7) with $2, 4, \ldots, 12$ flows. In this scenario, the aforementioned naive grid optimization runs quickly into computational problems, as a computation for 4 flows already took approximately a day. Therefore, we implemented the so called Pattern Search [28]. Here, a function is minimized by changing arguments only in a single direction. If multiple modifications lead to a descent, a step in the direction of all successful intermediate steps is attempted. The results of the ratio

$$\frac{\text{Computation time new method}}{\text{Computation time standard approach}}$$

for these experiments are depicted in Figure 9.

Under Pattern Search, we observe that computational overhead scales only linearly with the number of invocations of the Lyapunov inequality. This indicates that a good trade-off between cost and accuracy gain can be achieved, if optimization is done carefully.

V. Direct Application to Delay Bounds

At first glance, it is tempting to apply Lyapunov's inequality to the delay bound calculation as well. That is, we would modify the computation of the delay's violation probability as follows:

$$\begin{aligned} \mathrm{P}(d(t) > T) &\overset{(4)}{\leq} \mathrm{E}\left[e^{\theta (A \oslash S)\,(t+T,t)}\right] \\ &\overset{(2)}{=} \mathrm{E}\left[e^{\theta \max_{0 \leq i \leq t+T} \{A(i,t) - S(i,t+T)\}}\right] \\ &= \mathrm{E}\left[e^{\theta \max_{0 \leq i \leq t} \{A(i,t) - S(i,t+T)\}}\right] \end{aligned}$$

$$\overset{(9)}{=} \inf_{l\geq 1}\left\{\left(\mathrm{E}\left[e^{l\theta \max_{0\leq i\leq t}\{A(i,t)-S(i,t+T)\}}\right]\right)^{\frac{1}{l}}\right\}$$
$$\overset{(5)}{\leq} \inf_{l\geq 1}\left\{\left(\sum_{i=0}^{t}\mathrm{E}\left[e^{l\theta(A(i,t)-S(i,t+T))}\right]\right)^{\frac{1}{l}}\right\}, \tag{13}$$

where we used that $A(s,t)=0$ for $s\geq t$ in the third line and the quasi-Union bound in the last inequality. Owing to the fact that this estimates a probability, only values below 1 are of interest for (13). Disappointingly for this case, no improvement can be obtained, as the next theorem states.

Theorem 12. *Let a delay bound T according to* (13) *exist such that*

$$\sum_{i=0}^{t}\mathrm{E}\left[e^{l\theta(A(i,t)-S(i,t+T))}\right] < 1. \tag{14}$$

If l and θ are optimized (denoted by l^* and θ^*), then $l^*=1$, i.e., no improvement can be achieved.

Proof: Assume that l^* and θ^* are the optimal parameters for (13) and that $l^*>1$. This means that there exist $1\leq l'<l^*$ and $\theta'>\theta^*$ such that $l'\theta'=l^*\theta^*$. But this means

$$\left(\sum_{i=0}^{t}\mathrm{E}\left[e^{l^*\theta^*(A(i,t)-S(i,t+T))}\right]\right)^{\frac{1}{l^*}}$$
$$=\left(\sum_{i=0}^{t}\mathrm{E}\left[e^{l'\theta'(A(i,t)-S(i,t+T))}\right]\right)^{\frac{1}{l^*}}$$
$$>\left(\sum_{i=0}^{t}\mathrm{E}\left[e^{l'\theta'(A(i,t)-S(i,t+T))}\right]\right)^{\frac{1}{l'}},$$

where we inserted $l^*\theta^*=l'\theta'$ in the second line. In the third line, we used that $x^{\frac{1}{l^*}}>x^{\frac{1}{l'}}$ holds for all $x\in(0,1)$ and $l^*>l'\geq 1$. Clearly, this is a contradiction to our assumption that we had an optimal solution. Thus, the optimal l^* must be equal to 1. ∎

As a consequence, the Lyapunov approach can only indirectly decrease delay bounds via the output bound calculation. The same holds for the backlog bound (the proof follows along the same lines).

VI. Conclusion

In this paper, we proposed a novel approach to improve the MGF output bound calculation in the Stochastic Network Calculus using Lyapunov's inequality. We also gave a proof that shows why this is a valid bound and that it is always at least as accurate as the state-of-the-art method. It is also shown in comprehensive numerical evaluations that the delay's violation probability can be improved for two server topologies as well as fat trees. Our evaluation indicated a significant gain in some cases while leading to more moderate improvements on average. For a fat tree, we observed a very high gain as the number of cross flows is increased. These gains come conceptually for free, as no additional constraints have to be imposed, thus making our approach minimally invasive. Yet, from a computational perspective the gain comes at the price of a higher-dimensional optimization in the last stage of computing the bounds. Fortunately, our experiments indicate that the computational overhead only scales linearly with the invocations of the Lyapunov inequality under a carefully chosen optimization method.

Taking into account the crucial role of the output bound, we believe that we have made a significant contribution to the SNC network analysis. On the other hand, there are still many open challenges in the analysis of larger and more complex networks, e.g., dealing effectively with correlations in the traffic flows, which are left for future work.

References

[1] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs, "The internet at the speed of light," in *Proc. ACM Workshop on Hot Topics in Networks'14*, ser. HotNets-XIII, 2014, pp. 1–7.

[2] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[3] A. Whitmore, A. Agarwal, and L. Da Xu, "The internet of things–a survey of topics and trends," *Springer Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[4] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proc. ACM Design Automation Conference'10*, 2010, pp. 731–736.

[5] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

[6] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Transactions on information theory*, vol. 37, no. 1, pp. 114–131, 1991.

[7] ——, "A calculus for network delay, part II: Network analysis," *IEEE Transactions on information theory*, vol. 37, no. 1, pp. 132–141, 1991.

[8] C.-S. Chang, *Performance guarantees in communication networks*. London: Springer-Verlag, 2000.

[9] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. New York: Springer-Verlag, 2001.

[10] F. Ciucu, A. Burchard, and J. Liebeherr, "A network service curve approach for the stochastic analysis of networks," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'05)*, vol. 33, no. 1, 2005, pp. 279–290.

[11] M. Fidler, "An end-to-end probabilistic network calculus with moment generating functions," in *Proc. IEEE IWQoS'06*, Jun. 2006, pp. 261–270.

[12] Y. Jiang and Y. Liu, *Stochastic network calculus*. Springer, 2008, vol. 1.

[13] F. Ciucu and J. Schmitt, "Perspectives on network calculus – no free lunch, but still good value," in *Proc. ACM SIGCOMM'12 Conference*, New York, NY, USA, Aug. 2012, pp. 311–322.

[14] C. Li, A. Burchard, and J. Liebeherr, "A network calculus with effective bandwidth," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1442–1453, 2007.

[15] F. Ciucu, F. Poloczek, and J. Schmitt, "Sharp per-flow delay bounds for bursty arrivals: The case of FIFO, SP, and EDF scheduling," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'14)*, Toronto, Canada, 2014.

[16] A. Rizk, F. Poloczek, and F. Ciucu, "Computable bounds in fork-join queueing systems," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'15)*, vol. 43, no. 1, 2015, pp. 335–346.

[17] T. Zhu, D. S. Berger, and M. Harchol-Balter, "SNC-meister: Admitting more tenants with tail latency SLOs," in *Proc. ACM Symposium on Cloud Computing (SoCC'16)*, 2016.

[18] M. Fidler and Y. Jiang, "Non-asymptotic delay bounds for (k, l) fork-join systems and multi-stage fork-join networks," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'16)*, 2016, pp. 1–9.

[19] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus - design and evaluation of an accurate and fast analysis," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'17)*, 2017.

[20] F. Poloczek and F. Ciucu, "Scheduling analysis with martingales," *Elsevier Performance Evaluation*, vol. 79, pp. 56–72, 2014.
[21] ——, "Service-martingales: Theory and applications to the delay analysis of random access protocols," in *Proc. IEEE Conference on Computer Communications (INFOCOM'15)*, 2015, pp. 945–953.
[22] F. Ciucu, F. Poloczek, and J. Schmitt, "Stochastic upper and lower bounds for general markov fluids," in *Proc. IEEE International Teletraffic Congress (ITC 28'16)*, vol. 1, 2016, pp. 184–192.
[23] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch ..." in *Proc. IEEE International Conference on Computer Communications (INFOCOM'08)*, Phoenix, AZ, USA, Apr. 2008.
[24] M. A. Beck, "Advances in theory and applicability of stochastic network calculus," Ph.D. dissertation, TU Kaiserslautern, 2016.
[25] F. Ciucu, F. Poloczek, and J. B. Schmitt, "Sharp bounds in stochastic network calculus," *CoRR*, vol. abs/1303.4114, 2013. [Online]. Available: http://arxiv.org/abs/1303.4114
[26] R. Nelson, *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer, 1995.
[27] C. Courcoubetis and R. Weber, "Buffer overflow asymptotics for a buffer handling many traffic sources," *Journal of Applied Probability*, vol. 33, pp. 886–903, 1996.
[28] R. Hooke and T. A. Jeeves, ""Direct Search" Solution of Numerical and Statistical Problems," *Journal of the ACM (JACM)*, vol. 8, no. 2, pp. 212–229, 1961.

APPENDIX

A. Proof of Proposition 11

We have already seen in Subsection III-A that

$$\begin{aligned}&\mathrm{E}\left[e^{\theta A'(s,t)}\right]\\ \leq&\inf_{l\geq 1}\left\{\left(\sum_{i=0}^{s}\mathrm{E}\left[e^{l\theta(A(i,t)-S(i,s))}\right]\right)^{\frac{1}{l}}\right\},\end{aligned}$$

which can be continued with

$$\begin{aligned}&\inf_{l\geq 1}\left\{\left(\sum_{i=0}^{s}\mathrm{E}\left[e^{l\theta(A(i,t)-S(i,s))}\right]\right)^{\frac{1}{l}}\right\}\\ =&\inf_{l\geq 1}\left\{\left(\sum_{i=0}^{s}\mathrm{E}\left[e^{l\theta A(i,t)}\right]\mathrm{E}\left[e^{-l\theta S(i,s)}\right]\right)^{\frac{1}{l}}\right\}\\ \leq&\inf_{l\geq 1}\left\{e^{\theta(\sigma_A(l\theta)+\sigma_S(l\theta))}\right.\\ &\left.\cdot\left(\sum_{i=0}^{s}e^{l\theta(\rho_A(l\theta)(t-i)+\rho_S(l\theta)(s-i))}\right)^{\frac{1}{l}}\right\},\end{aligned}$$

where we, again, used the independence of arrivals and service in the second line and the $(\sigma(\theta),\rho(\theta))$-constraints for arrivals and service in the third line.

Since we assume that $\rho_A(l\theta) < -\rho_S(l\theta)$, we obtain by convergence of the geometric series

$$\begin{aligned}\cdots=&\inf_{l\geq 1}\left\{e^{\theta(\rho_A(l\theta)(t-s)+\sigma_A(l\theta)+\sigma_S(l\theta))}\right.\\ &\left.\cdot\left(\sum_{j=0}^{s}e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))j}\right)^{\frac{1}{l}}\right\}\\ \leq&\inf_{l\geq 1}\left\{e^{\theta(\rho_A(l\theta)(t-s)+\sigma_A(l\theta)+\sigma_S(l\theta))}\right.\\ &\left.\cdot\left(\frac{1}{1-e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))}}\right)^{\frac{1}{l}}\right\}.\end{aligned}$$

This finishes the proof, as this is equal to

$$\begin{aligned}\cdots=&\inf_{l\geq 1}\left\{e^{\theta(\rho_A(l\theta)(t-s)+\sigma_A(l\theta)+\sigma_S(l\theta))}\right.\\ &\left.\cdot e^{\theta\left(-\frac{1}{l\theta}\log\left(1-e^{l\theta\left(\rho_A(l\theta)+\rho_S(l\theta)\right)}\right)\right)}\right\},\end{aligned}$$

which yields

$$\begin{aligned}\sigma_{A'}(\theta)=&\sigma_A(l\theta)+\sigma_S(l\theta)-\frac{1}{l\theta}\log\left(1-e^{l\theta(\rho_A(l\theta)+\rho_S(l\theta))}\right),\\ \rho_{A'}(\theta)=&\rho_A(l\theta)\end{aligned}$$

as the theorem states.

Hierarchical Layer Selection with Low Overhead in Prioritized Network Coding

Marie Schaeffer, Roman Naumann, Stefan Dietzel, and Björn Scheuermann
Humboldt-Universität zu Berlin, Germany
Email: {marie.schaeffer, roman.naumann, stefan.dietzel}@hu-berlin.de, scheuermann@informatik.hu-berlin.de

***Abstract*—Network coding simplifies routing decisions, improves throughput, and increases tolerance against packet loss. A fundamental limitation, however, is delay: decoding requires as many independent linear combinations as data blocks. Prioritized network coding reduces this delay problem by introducing a hierarchy of prioritization layers. What remains is the problem of choosing a layer to approach two often-contradicting goals: reduce delay until prioritized layers can be decoded and keep the total number of transmissions low. In this paper, we propose an algorithm for this problem that – based on limited feedback – primarily minimizes per-layer delay but identifies opportunities to reduce the required transmissions when per-layer delay is unaffected. Our evaluation shows that our algorithm improves per-layer delay compared to hierarchical network coding and is close to the theoretical optimum number of total transmissions.**

I. Introduction

Network coding (NC) is a widely studied approach to communication systems [1]. Originally introduced by Ahlswede *et al.* [2] as a technique to improve the throughput in networks, NC has been proven to benefit many fields since, e. g., peer-to-peer applications [3]–[5], network streaming [6], and combinations thereof [7]. NC also improves robustness, which means that the system better copes with packet loss. Distributed algorithms can be simplified, and link capacities can be saturated more effectively [1]. To implement these benefits, NC breaks with traditional routing paradigms. Namely, nodes combine two or more incoming packets and send these newly built combinations instead of just forwarding the original packets. In this paper, we discuss an improvement for the most broadly studied class of network coding, linear network coding, where original packets are combined into *linear* combinations [8].

One restriction inherent to NC is that it introduces additional delay. With high probability, a receiver cannot retrieve any of the original content as long as the number of received linear combinations is lower than the number of messages that were combined [9], [10]. Prioritized network coding (PNC) [11] builds on linear network coding and reduces decoding delay by introducing a hierarchy of priority layers on the original messages. Linear combinations are computed per priority layer rather than using the full message set. Consequently, a receiver is able to decode a prioritized subset of messages with fewer linear combinations. This encoding technique is also known as expanding window random linear coding [12]. The order in which different layers' linear combinations are sent has a direct impact on individual layers' achievable decoding performance and principal decodability at any given point in time [13], [14]. In order to achieve a reduced delay and at the same time avoid significant additional overhead, combinations have to be built and sent in a sequence that reflects the layers' individual priorities.

ISBN 978-3-903176-08-9

In summary, PNC can reduce per-message delay, but it only does so under the right circumstances: all senders need to carefully choose the correct layers for use in their next linear combination. Otherwise, linear combinations either have an increased chance of being linearly dependent, which results in increased message overhead, or they introduce additional per-packet decoding delay, which results in less effective prioritization. This selection problem of PNC has not yet been studied in detail for scenarios with limited knowledge about the receivers' decoder state. Existing approaches select layers uniformly at random (e. g., [11], known as hierarchical network coding (HNC)), or they use a weighted random choice, giving higher weight to prioritized layers [12]. Both strategies cause overhead, since they result in an increased probability of non-innovative content being sent.

In this paper, we take a more structured approach to the selection problem and analyze – based on limited feedback messages –, which selection of layers (a) reduces the total number of transmissions and (b) improves per-message decoding delay. We derive performance indicators that allow a node to classify layer choices based on these two criteria and propose an algorithm, eNhanced Prioritized nEtwork Coding (iNsPECt), which implements a deterministic strategy for choosing the layer that is used to generate the next linear combination. Based on these performance indicators, we instantiate a wireless multi-hop protocol that is based on single-hop feedback messages, which concisely summarize each node's decoder matrix state. Our protocol thereby implements prioritization with much less overhead than existing approaches while retaining the low decoding delay.

The remainder of this paper is structured as follows. Section II discusses related work, and Section III introduces our system model. In Section IV, we approach the layer selection problem with two performance indicators that guide layer choices. We further introduce an algorithm that yields optimal results under an analytical model, which we instantiate as a practical network protocol in Section V. Our simulative evaluation compares the protocols iNsPECt, HNC, and NC in Section VI before Section VII concludes the work.

II. Related Work

NC was introduced by Ahlswede *et al.* [2] to improve throughput in communication networks. In their work, the network model is a directed graph with one node as the source and multiple nodes as receivers. Ahlswede *et al.* demonstrate that the optimal throughput, which is given by the "minimum cut" between the source node and any receiver in a network graph, can be achieved when the nodes send linear combinations of the original messages. Later, Ho *et al.* [9] showed that randomly chosen linear coefficients $c_1, c_2, \dots$ over a finite field $\mathbb{F}_q$ are sufficient to achieve optimal flow rates; this approach is called random linear network coding (RLNC). With RLNC, linear combinations are built by multiplying the n original messages $m^{(1)}, m^{(2)}, \dots, m^{(n)}$ with the random coefficients, the j-th linear combination $X^{(j)}$ being:

$$\boldsymbol{X}^{(j)} = \sum_{i=1}^{n} c_i^{(j)} \boldsymbol{m}^{(i)}. \tag{1}$$

When multiple such combinations are received, they form a system of linear equations. The original messages can be retrieved by solving the system with, for example, Gaussian elimination (GE), once sufficient combinations have been received. In general, it is not possible to decode a subset of messages with fewer than n linear combinations. However, all messages can be decoded immediately once enough linear combinations were received. This has been described as the "all-or-nothing property" [15].

As Nguyen *et al.* [11] note, for many applications, NC's delay is not tolerable. Consequently, Nguyen *et al.* propose PNC, which is based on RLNC and reduces per-packet delay. PNC introduces hierarchical layers $R_1, R_2, \dots, R_{|\boldsymbol{R}|}$ of prioritized packets. That is, linear combinations of the l-th layer encode only messages from $m^{(1)}, m^{(2)}, \dots, m^{(R_l)}$:

$$\boldsymbol{x}^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} \boldsymbol{m}^{(i)}, \quad \text{for a layer-}l\text{ combination.} \tag{2}$$

An important question is how to determine which layer to choose for generating new linear combinations. The most basic approach, used by HNC [11], is to choose layers uniformly at random. HNC generally provides lower per-packet delay than RLNC, but increases the overhead due to non-informative linear combinations, i. e., linear combinations from layers that can already be decoded.

Esmaeilzadeh *et al.* [14] explicitly studied the layer selection problem both for systems without any knowledge about the receivers' decoder states and for systems with perfect knowledge about the decoder states. The proposed layer selection algorithm is based on an exhaustive search through packet erasures. In addition, finite-horizon Markov decision processes are proposed for the perfect-knowledge system model. The authors describe their perfect-knowledge model as idealistic, since perfect knowledge is usually unavailable. Additionally, both algorithms' high computational complexity makes them unusable for practical applications with greater numbers of layers and/or users, but they may serve as a theoretical upper bound. We, different to [14], assume *limited* knowledge of the neighbors' decoder states and derive a simpler performance indicator that does not require exhaustive searching.

Naumann *et al.* [13] denote that all efficient layer selection schemes for PNC will send linear combinations roughly in order of priority, which they exploit to implement specialized Gaussian-elimination-based decoders. Such decoders improve both memory footprint and computational decoding complexity by reordering rows and inverting certain GE elimination steps in a joint decoder matrix for all layers. Our approach is a natural fit for such decoders, as it is based on a greedy strategy that sends in order of prioritization most of the time. Even more so, we provide an upper bound on the limit of deviation from this greedy strategy so that the asymptotic bounds on computational decoding overhead described in [13] hold.

Shenglan Huang *et al.* [16] build upon HNC with uniform random layer selection to minimize the amount of redundant packets sent in a multi-sender use case. Their algorithm estimates, according to loss rates and information about links, the ideal number of linear combinations that each sender should produce to reduce linearly dependent combinations. The algorithm does not, however, provide layer selection capabilities different from HNC.

Chau *et al.* [17] also use the HNC coding technique but propose an additional coding scheme that combines messages from more than one HNC-coded generation. Thereby, the scheme provides additional redundancy, which protects against packet loss, and reduces the number of transmissions until all layers can be decoded. Our proposed layer selection technique could be used in conjunction with their HNC-based coding scheme, since it does not modify the coding format.

Approaches different from hierarchical PNC have been proposed to reduce per-packet delay in NC: Shrader *et al.* [18], for example, propose to employ a systematic coding approach. Namely, a subset of the network's nodes sends uncoded packets in some circumstances. Due to the selection of nodes, the level of error protection is not reduced, but the non-encoded packets reduce per-packet delay as they do not require decoding. Yan *et al.* [15] instead trade correctness of decoded information for an increased chance of rank-deficient decoding by regarding the decoder matrix as a collection of underdetermined systems and implementing rank-deficient decoders. Finally, Claridge *et al.* [19] demonstrate that rank deficient decoding without chance of error is feasible when using a small enough finite field. Such a small field, however, also reduces the level of error protection and increases the chance of linear dependency.

III. System Model

A. Information model

We assume that the information to be transmitted by a source node can be split into equally sized messages $\boldsymbol{M} = (\boldsymbol{m^{(1)}}, \boldsymbol{m^{(2)}}, \dots, \boldsymbol{m^{(n)}})$. Each message $\boldsymbol{m^{(i)}}$ consists of b symbols $(m_1^{(i)}, m_2^{(i)}, \dots, m_b^{(i)})$ over a finite field $\mathbb{F}_q$. We are concerned with prioritized messages, i. e., some messages convey more information than others. In the following, and

w. l. o. g., we assume $\boldsymbol{m^{(i)}}$ has higher priority than $\boldsymbol{m^{(j)}}$ for all $1 \leq i < j \leq n$. PNC introduces hierarchical layers that we formalize as the vector $\boldsymbol{r} = (r_1, r_2, \ldots, r_{|\boldsymbol{r}|}) \in \mathbb{N}^{|\boldsymbol{r}|}$. Each entry r_l denotes the number of messages that the layer l adds, so it holds that $n = \sum_{l=1}^{|\boldsymbol{r}|} r_l$. Analogously, we define $\boldsymbol{R}$ as the cumulated layer vector with $R_i = \sum_{l=1}^{i} r_l$.

To transmit information, each source creates a sequence of network-coded packets; the j-th packet has the form $(\boldsymbol{c^{(j)}}, \boldsymbol{x^{(j)}})$. Here, $\boldsymbol{c^{(j)}}$ is called the encoding vector and consists of randomly chosen coefficients $(c_1^{(j)}, c_2^{(j)}, \ldots, c_n^{(j)})$, i. e., random symbols over $\mathbb{F}_q$. The second component, $\boldsymbol{x^{(j)}}$, is called the information vector and contains the actual linear combination. An information vector of the l-th ($1 \leq l \leq |\boldsymbol{r}|$) layer and j-th coded packet is encoded as follows [1], [12]:

$$x_k^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} m_k^{(i)} \quad \forall 1 \leq k \leq b \tag{3}$$

Since finite field operations are generally performed over all symbols of a message or information vector, we usually omit the symbol index k, which reduces Equation (3) to Equation (2). Multiple generations or multiple source nodes fit the above definitions by executing the encoding mechanism repeatedly in sequence or in parallel, respectively.

The finite field $\mathbb{F}_{2^8}$ is a typical choice for network coding [1], [20] and is used in the following. With $\mathbb{F}_{2^8}$, linear dependencies between randomly generated combinations are unlikely [9], and the elements' binary representations occupy one byte each, which is advantageous in practical systems.

B. Network model

Our network model is a wireless network with multiple nodes. Nodes transmit information as broadcast messages. Such messages may be received or lost by multiple nodes independently. We allow several nodes to be source nodes, which generate new messages $\boldsymbol{M}$. Both source nodes and non-source nodes receive, re-encode, and transmit information.

Before any transmission starts, a source node holds the complete data set, which is to be transmitted via PNC to the sink nodes. To simplify the system model, we assume in this work that all nodes in the network are sink nodes, which is a typical simplification for network coding. It alleviates the need to share topology information and improves the network capacity utilization [21]. The generality of our results is not affected by this simplification: if all nodes within the network receive the data, then any single sink or few sinks trivially have the information, too. As a consequence, a source node of one transmission is a sink node to other nodes, as well.

C. Problem statement

According to our information model, data to be transmitted is partitioned into different priority layers. Given this partition, the problem statement is to find an *efficient* transmission process that implements the prioritization scheme that is defined by the layers. Here, "efficient" comprises two aspects: low per-layer delay and low number of total transmissions. The per-layer delay for prioritized layers describes the prioritization performance, whereas the total number of transmissions describes the overhead that is introduced. Ideally, both aspects are jointly optimized by *implementing effective prioritization without introducing overhead.*

It is impossible, however, to achieve low delay at the same time as low overhead in all situations, as these goals may conflict. Rather, we propose a protocol that jointly optimizes both whenever possible and prioritizes low delay in conflict situations. To understand the connection between delay and total transmission number, consider an example topology with one source node S and three non-source nodes N_1, N_2, and N_3. Assume a simple PNC system with a generation size $n = 4$ and two priority layers $\boldsymbol{r} = (2, 2)$. We will now discuss two scenarios: one where delay and overhead are in conflict and one where both can jointly be optimized.

As the first scenario, assume nodes N_1, N_2, and N_3 have received $1, 1$, and 0 linearly independent combinations of the first layer, respectively, and only N_3 has received 1 linear combination of the second layer. Now, S sends two more linear combinations of the first layer. Then, N_1 and N_2 can decode the first layer after having received the first linear combination, and node N_3 can decode after having received both linear combinations. However, two more linear combinations of the second layer must be sent before all nodes can retrieve the second priority layer. If, instead, S immediately starts sending linear combinations of the second layer, all nodes must wait for one more transmission before they can decode the first layer. After only three transmissions in total (instead of four, as before), all nodes can decode layer one *and* two.

As the second scenario, consider nodes N_1, N_2, and N_3 have initially received $2, 1$, and 0 independent combinations of the first layer and $1, 2, 3$ independent linear combinations of the second layer, respectively. In this case, sending only one combination of the (lower priority) second layer from the beginning saves one transmission *and* achieves minimal per-layer delay.

The important observation here is that the goals of prioritization and not introducing extraneous transmissions *can* conflict, but this is *not always* the case. Our goal is to provide optimal prioritization first, but identify such occasions where we can save transmissions without introducing per-layer delay.

IV. Proposed Algorithm

In this section, we approach the problem statement with a simplified, theoretical model: nodes have knowledge of their neighbors' decoder matrix rank and each layer's linear subspace dimension. There are no packet losses or delays. We describe the proposed algorithm, iNsPECt, from the perspective of an individual node. In Section V, we incorporate packet losses, delays, and the need for feedback messages in the design of a network protocol based on this algorithm.

Our proposed algorithm combines two complementary strategies, which we term "Ord" and "SL," that are used to decide which layer to select for transmission of the next linear combination. Both strategies (and our algorithm) make

use of the fact that when one node's decoder matrix for a given layer has a higher rank than its neighbor's, sending a linear combination is with high probability innovative. "Ord," short for *in order,* is a greedy strategy that selects layers in strict order of prioritization; "SL" sends linear combinations of a *single layer* only. Our key idea is to use strategy Ord by default to minimize delay for prioritized layers. But we resort to sending a lower priority layer (with strategy SL) if it reduces the required total transmissions and does not negatively affect per-layer delay. Strategy SL takes a target layer i as a parameter; SL(i) sends only linear combinations of layer i until layer i – and thus all higher priority layers, as well – can be decoded. Obviously, SL(i) requires the minimum number of transmissions until layer i can be decoded; and strategy SL($|\boldsymbol{r}|$) equals RLNC. Strategy Ord instead sends linear combinations of the highest priority non-decodable layer until each neighbor can decode that layer. It then continues with the next layer. Therefore, Ord ensures that high priority layers are always decoded before lower-priority layers.

To determine which strategy to use, our algorithm models the benefits of choosing one strategy over the other at any point in time with two performance indicators. Each indicator takes a parameter i and returns the benefits or drawbacks that result from choosing strategy SL(i) over Ord.

The first indicator, $Q_{\text{rt}}(i)$, counts the *savings in total number of transmissions* until each neighbor of a node can decode layer i. Positive values indicate that using SL(i) is beneficial over choosing Ord. The second indicator, $Q_{\text{dc}}(i)$, analogously counts the *additional per-layer delay* (in transmissions) until a node's neighbors can decode layers 1 to i, cumulated over the layers and all neighbors. Positive values indicate additional overhead introduced by choosing SL(i). In the following, we first derive the two indicators from our simplified system model and then define the selection algorithm.

A. Performance indicators

1) Reduction in transmissions: We define $\text{RT}_{\text{SL}}^{(*)}(i)$ as the number of required transmissions until all neighbors can decode layer i using the SL strategy. Analogously, $\text{RT}_{\text{Ord}}^{(*)}(i)$ denotes the number of required transmissions using the Ord strategy. Consequently, the savings in transmissions are:

$$Q_{\text{rt}}(i) = \text{RT}_{\text{Ord}}^{(*)}(i) - \text{RT}_{\text{SL}}^{(*)}(i). \tag{4}$$

Next, we derive $\text{RT}_{\text{Ord}}^{(*)}(i)$ and $\text{RT}_{\text{SL}}^{(*)}(i)$. Let $\gamma_l^{(x)}$ be the number of independent linear combinations of layer l that neighbor x has received (and equivalently, the number of dimensions of the linear subspace that pertains to layer l), and let $\Gamma_l^{(x)}$ be the accumulated number of received combinations from layer 1 to l of neighbor x. Let $\text{RT}_{\text{SL}}^{(x)}(i)$ be the number of transmissions required for a single node x to decode layer i. Layer i can be decoded once layer i's linear subspace has full rank, i. e., when $\Gamma_i^{(x)} = R_i$. Alternatively, layer i may be decoded when a lower priority subspace has full rank, i. e., $\Gamma_j^{(x)} = R_j$ for some $j > i$. Thus,

$$\text{RT}_{\text{SL}}^{(x)}(i) = \min\Big(R_i - \Gamma_i^{(x)}, \min_{j=i+1}^{|\boldsymbol{r}|}\big(R_j - \Gamma_j^{(x)}\big)\Big)$$

$$\Leftrightarrow \text{RT}_{\text{SL}}^{(x)}(i) = \min_{j=i}^{|\boldsymbol{r}|}\big(R_j - \Gamma_j^{(x)}\big). \tag{5}$$

To generalize $\text{RT}_{\text{SL}}^{(x)}(i)$ to $\text{RT}_{\text{SL}}^{(*)}(i)$, we take the maximum over all neighbors:

$$\text{RT}_{\text{SL}}^{(*)}(i) = \max_{x\,\in\,\text{Neigh.}} \text{RT}_{\text{SL}}^{(x)}(i). \tag{6}$$

We construct $\text{RT}_{\text{Ord}}^{(*)}(i)$ recursively. Since strategies Ord and SL are identical for $i = 1$, it holds that

$$\begin{aligned} &\text{RT}_{\text{Ord}}^{(x)}(1) = \text{RT}_{\text{SL}}^{(x)}(1), \text{ and} \\ &\text{RT}_{\text{Ord}}^{(*)}(1) = \max_{x\,\in\,\text{Neigh.}} \text{RT}_{\text{SL}}^{(x)}(1) = \text{RT}_{\text{SL}}^{(*)}(1). \end{aligned} \tag{7}$$

Counting the required transmissions for the $(i+1)$-th layer, we first count the transmissions from the i-th layer and then add the remaining, maximum missing matrix rank over all neighbor nodes:

$$\begin{aligned} &\text{RT}_{\text{Ord}}^{(*)}(i+1) = \\ &\qquad \text{RT}_{\text{Ord}}^{(*)}(i) + \max_{x\,\in\,\text{Neigh.}}\big(\text{RT}_{\text{SL}}^{(x)}(i+1) - \text{RT}_{\text{SL}}^{(x)}(i)\big), \end{aligned} \tag{8}$$

which, if we define $\text{RT}_{\text{SL}}^{(x)}(0) = 0$, reduces to

$$\text{RT}_{\text{Ord}}^{(*)}(i) = \sum_{j=0}^{i-1} \max_{x\,\in\,\text{Neigh.}}\big(\text{RT}_{\text{SL}}^{(x)}(j+1) - \text{RT}_{\text{SL}}^{(x)}(j)\big). \tag{9}$$

2) Per-layer delay: Analogously, we define $\text{DC}_{\text{SL}}^{(x)}(i)$ and $\text{DC}_{\text{Ord}}^{(x)}(i)$ as the cumulative per-layer delay (in transmissions) until node x can decode each layer up to i with the SL and Ord strategies, respectively. Similarly, $\text{DC}_{\text{SL}}^{(*)}(i)$ and $\text{DC}_{\text{Ord}}^{(*)}(i)$ define this delay cumulatively for all neighbor nodes. Our indicator,

$$Q_{\text{dc}}(i) = \text{DC}_{\text{SL}}^{(*)}(i) - \text{DC}_{\text{Ord}}^{(*)}(i), \tag{10}$$

gives the additional per-layer delay that results from choosing strategy SL over Ord.

With the SL strategy, each node waits a timespan that is independent from the other nodes' decoder matrix states, as each node's rank of layer i increases independently until full rank is obtained. As the SL strategy only sends linear combinations of layer i, each non-decodable layer below i of neighbor x will become decodable after exactly $\text{RT}_{\text{SL}}^{(x)}(i)$ transmissions. Therefore, we count the number of non-decodable layers (right-hand factor) multiplied by the number of transmissions required for decoding each layer (left-hand factor):

$$\text{DC}_{\text{SL}}^{(x)}(i) = \text{RT}_{\text{SL}}^{(x)}(i) \cdot \sum_{k=1}^{i} \min\big(\text{RT}_{\text{SL}}^{(x)}(k), 1\big), \tag{11}$$

$$\text{DC}_{\text{SL}}^{(*)}(i) = \sum_{x\,\in\,\text{Neigh.}} \text{DC}_{\text{SL}}^{(x)}(i). \tag{12}$$

Again, we derive the per-layer delay for the Ord strategy recursively and in two steps. First, we derive the non cumulated per-layer delay $\text{NC}_{\text{Ord}}^{(x)}(i)$ so that

$$\text{DC}_{\text{Ord}}^{(x)}(i) = \sum_{j=1}^{i} \text{NC}_{\text{Ord}}^{(x)}(j). \tag{13}$$

Again, for $i = 1$ both strategies behave identically, thus $\mathrm{DC}^{(x)}_{\mathrm{Ord}}(1) = \mathrm{NC}^{(x)}_{\mathrm{Ord}}(1) = \mathrm{DC}^{(x)}_{\mathrm{SL}}(1)$ and $\mathrm{DC}^{(*)}_{\mathrm{Ord}}(1) = \mathrm{DC}^{(*)}_{\mathrm{SL}}(1)$. For $i+1$, we distinguish between two cases based on whether the $(i+1)$-th layer can be decoded if the i-th layer can be decoded, which formally is the proposition:

$$\mathrm{RT}^{(x)}_{\mathrm{SL}}(i+1) = \mathrm{RT}^{(x)}_{\mathrm{SL}}(i). \tag{14}$$

If eq. (14) holds, the induction step is trivial, as no additional delay comes from layer $i+1$: $\mathrm{NC}^{(x)}_{\mathrm{Ord}}(i+1) = \mathrm{NC}^{(x)}_{\mathrm{Ord}}(i)$. If eq. (14) does not hold, node x requires exactly as many independent linear combinations as it is short of full rank to decode layer $i+1$, i. e., $\mathrm{RT}^{(x)}_{\mathrm{SL}}(i+1) - \mathrm{RT}^{(x)}_{\mathrm{SL}}(i)$. Since the Ord strategy will not start sending linear combinations of rank $i+1$ until *all* other neighbors can decode layer i, we also have to wait for $\mathrm{RT}^{(*)}_{\mathrm{Ord}}(i)$ transmissions before the $(i+1)$-th layer's rank increases:

$$\mathrm{NC}^{(x)}_{\mathrm{Ord}}(i+1) = \begin{cases} \mathrm{NC}^{(x)}_{\mathrm{Ord}}(i), & \text{if eq. (14) holds, else} \\ \mathrm{RT}^{(*)}_{\mathrm{Ord}}(i) + \mathrm{RT}^{(x)}_{\mathrm{SL}}(i+1) - \mathrm{RT}^{(x)}_{\mathrm{SL}}(i). \end{cases} \tag{15}$$

For all nodes, we obtain:

$$\mathrm{DC}^{(*)}_{\mathrm{Ord}}(i) = \sum_{x \in \text{ Neigh.}} \mathrm{DC}^{(x)}_{\mathrm{Ord}}(i). \tag{16}$$

B. Algorithm

We have defined the two performance indicators $Q_{\mathrm{rt}}(i)$, which counts the required transmissions until layer i can be decoded by all neighbor nodes, and $Q_{\mathrm{dc}}(i)$, which counts the per-layer delay over all neighbors and layers up to i. We now use these indicators in an algorithm that takes a node's state as input and returns the layer choice as output. Whenever a node generates and transmits a linear combination, it executes the algorithm iNsPECt first, which is given in Figure 1.

To keep computational overhead low in practical systems, iNsPECt introduces a system parameter k_{ahead}, which bounds the number of layers that a node may deviate from the Ord strategy. For large numbers of layers, bounding the deviation with k_{ahead} not only improves our algorithm's performance, but allows to use optimized decoding techniques [13].

In Figure 1 line 1, the Ord strategy is employed by default. That is, a node selects the layer with the highest priority that any of its neighbors cannot decode. To reduce the total number of transmissions, our algorithm uses the previously defined performance indicators in two steps:

(1) check if a lower priority layer can save transmissions by computing Q_{rt} and
(2) check if the lower priority negatively affects per-layer delay by computing Q_{dc}.

These steps are repeatedly performed in the conditional at line 4 for all k_{ahead} candidate layers in the loop at lines 3 to 8. Whenever a candidate layer in the loop further reduces transmissions compared to the last candidate *and* does not increase per-layer delay, it is selected as next candidate. Finally, the selected layer is returned in line 9.

Input: for each neighbor $x \in$ Neigh., $\boldsymbol{\gamma}^{(x)}$ and $\boldsymbol{\Gamma}^{(x)}$
Output: layer index
Have: $n, \boldsymbol{r}, \boldsymbol{R}$, and system parameter k_{ahead}

1: choice $\leftarrow$ start $\leftarrow$ first non decodable layer of Neigh.
2: $\mathrm{last}_{\mathrm{rt}} \leftarrow Q_{\mathrm{rt}}(start)$
3: **for** $i \leftarrow \mathrm{start}+1, \ldots, \min(\mathrm{start}+k_{\mathrm{ahead}}, |\boldsymbol{r}|)$ **do**
4: **if** $Q_{\mathrm{rt}}(i) > \mathrm{last}_{\mathrm{rt}} \wedge Q_{\mathrm{dc}}(i) \leq 0$ **then**
5: choice $\leftarrow i$
6: $\mathrm{last}_{\mathrm{rt}} \leftarrow Q_{\mathrm{rt}}(i)$
7: **end if**
8: **end for**
9: **return** choice

Fig. 1. iNsPECt.

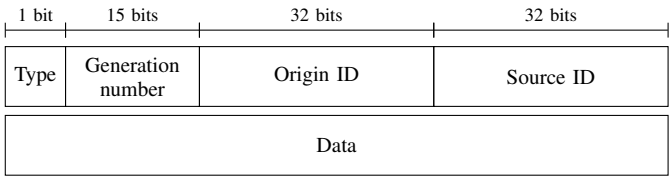


Fig. 2. Message format: general header and message specific data part.

Dependent on m, the number of neighbors from which feedback has recently been received, the algorithm's runtime is in $\mathcal{O}(m\, k_{\mathrm{ahead}}\, |\boldsymbol{r}|)$. For three layers, a common choice in multimedia streaming [11], the runtime is linear in the number of neighbors.

V. Protocol Design

We now describe a simple yet effective network protocol that instantiates the layer selection algorithm for practical systems. We describe the protocol for a single network coding generation and a single source. The protocol is executed in parallel when multiple sources exist in the network, and it is run repeatedly for subsequent generations.

A. Message types

The protocol requires only two types of messages: data messages, which contain linear combinations, and feedback messages, which concisely encode a node's decoder state. We use UDP as the underlying transport protocol, because reliability is ensured by network coding's forward error correction properties.

The general message format is shown in Figure 2: a leading bit is used to denote message type, and the remaining 15 bits of the first two octets encode the generation number that the message belongs to. Origin node and source node, each encoded using 32 bits, are used to manage neighbor state and assign linear combinations to the correct decoding system. Each node is assigned a unique number in the system; alternatively, the IP addresses can be used for local topologies.

Data messages contain linear combinations, which consist of an encoding vector and an information vector, as described in Section III-A. If a linear combination from a layer with

higher priority is sent, not all coefficients in the encoding vector are used; in that case, the remaining coefficients are set to the additive identity (i. e., "zeroes") of the finite field. Thus, encoding vectors contain n coefficients, whereas the information vector consists of b symbols. Each coefficient and symbol are elements of the finite field F_{2^8} and can be encoded as a single byte. Data messages, therefore, have a length of $b + n + 10$ bytes.

Feedback messages encode $\boldsymbol{\gamma}^{(x)}$; that is, they encode how many linearly independent combinations of each layer a node x has received. This is identical to the rank of each layer's decoder matrix or the dimension of each layer's linear subspace. A feedback message encodes each rank with two bytes; thus, the total feedback message length is $2 \cdot |\boldsymbol{r}| + 10$ bytes, where $|\boldsymbol{r}|$ is the total number of layers. Since the feedback messages' size does not depend on the generation size n nor on the chunk size b, feedback messages are usually much smaller than data messages.

B. Transmission mechanism

We employ a constant-rate approach to sending linear combinations. That is, every data message transmission interval λ_{data}, a linear combination is built according to Equation (3), encoded as a data message, and sent when the generation is not marked as fully transmitted. First, the iNsPECt algorithm is executed to determine the ideal layer i for building the next linear combination. If that layer-i's decoder matrix has insufficient rank to generate a linear combination, i is incremented repeatedly until a linear combination can be built. If, initially, no feedback is available, we default to sending a linear combination of the highest priority layer.

Whenever a layer-i linear combination is sent, the feedback vector $\boldsymbol{\gamma}^{(x)}$ for each neighbor x is incremented, presuming the linear combination's successful reception. In addition, a flag is set that indicates that the feedback vector is *assumed* instead of authoritative. If an assumed feedback vector indicates the generation is fully transmitted, the node continues to send linear combinations until an authoritative feedback message is received as confirmation. Thereby, nodes avoid delays at the end of each generation. By handling assumed and authoritative feedback in this way, we ensure that layers of lower-than-ideal priority may be selected, but never layers of higher priority. This bias may lead to increased per-layer delay for the current layer. It does not, however, cause additional overhead from linearly dependent combinations, nor does it affect the lower priority layers' decoding delay.

On reception of a linear combination, a node first counts the trailing number of additive identity elements in the encoding vector to determine the combination's layer. Next, the linear combination is inserted into each decoder matrix that pertains to a lower or equal priority than the linear combination's layer. Whether the newly received linear combination is innovative is determined using GE. If the rank of the matrix increases, the combination was innovative; otherwise, the new row is reduced to additive identity elements [1].

C. Feedback mechanism

Feedback is broadcast periodically and cumulatively for a generation's layers: once every feedback interval λ_{fb}, a feedback message is created for each incomplete generation. Usually, feedback messages are only sent when the generation is incomplete, i. e., the node's decoder matrix state does not have full rank for all layers. If, however, a linear combination for a complete generation is received, a feedback message that indicates successful reception of the whole generation (with $\Gamma_{|\boldsymbol{r}|} = R_{|\boldsymbol{r}|}$) is sent once.

The feedback's purpose is not only to inform other nodes about the current decoder state, but it also allows other nodes to learn about their neighborhood. When a node receives a feedback message from a node x, it includes x in its neighbor set (Neigh. in Figure 1) and updates $\boldsymbol{\gamma}^{(x)}$ with the rank vector that is included in the feedback message. If a feedback message is received where the combination of origin identifier and generation number is unknown, that message is ignored, as the feedback contained is not helpful. Feedback vectors and neighborhood states expire after a timeout that should be chosen as a multiple of the feedback interval to avoid incomplete neighbor sets.

The proportion between the data interval λ_{data} and λ_{fb} is important for the performance of the proposed algorithm. The smaller the feedback interval, the better each node's stored feedback represents its neighbors' decoder state, since it is updated more often. On the other hand, even tough feedback messages have a small size, more frequent feedback means more network capacity is used for traffic that does not directly contribute to the delivery of the sources' information.

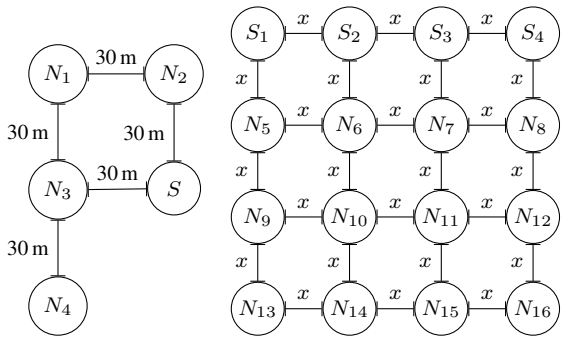
VI. Evaluation

We compare iNsPECt to PNC's HNC variant and RLNC, which we both described in Section II. As a *lower bound* on decoding delay, we additionally show the decoding time of the first layer, which results from sending only linear combinations of the first layer. We evaluate all protocols in three scenarios: a small-scale topology with a single source and small generations that comprise few source messages to evaluate the impact of varying feedback rates, a larger topology to show multi-hop capabilities and support for multiple sources, and finally a set of larger, randomized topologies to support the generality of our results.

A. Methodology

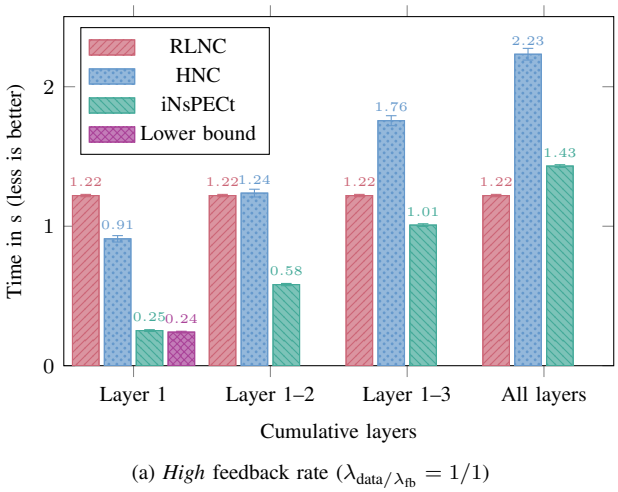
We evaluate using the discrete event network simulator ns-3 (version 3.25) [22]. Wireless links between nodes are modeled via YANS Wifi model [23] with 802.11g MAC and 2.4 GHz PHY. We simulate the physical channel with the log-distance propagation loss model[1] and the Rayleigh fast fading model, one superimposed on the other [24]. Nodes send actual linear combinations in the simulation, so there is a (small) chance for linear dependency even if a layer's decoder matrix does

[1]We choose the path-loss exponent $\gamma = 3.0$ and configured path loss at the reference distance 1 m according to Friis' model for 2.4 GHz, which is in line with a range of office and industrial environments [24], [25].

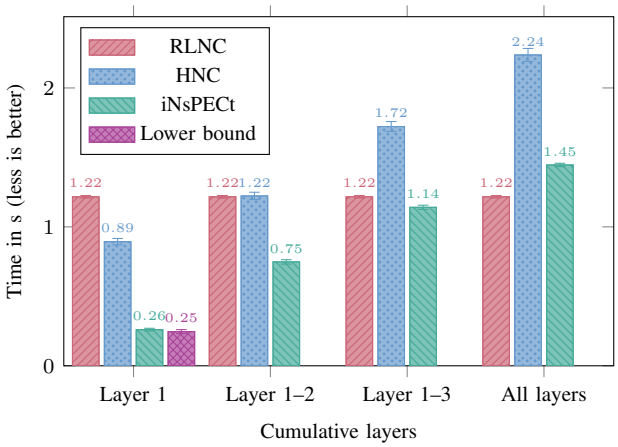


(a) Small scale topology (b) Larger network topology

Fig. 3. Simulated topologies.



(a) *High* feedback rate ($\lambda_{\text{data}}/\lambda_{\text{fb}} = 1/1$)



(b) *Low* feedback rate ($\lambda_{\text{data}}/\lambda_{\text{fb}} = 1/3$)

Fig. 4. Small topology results: per-layer delay for different feedback rates.

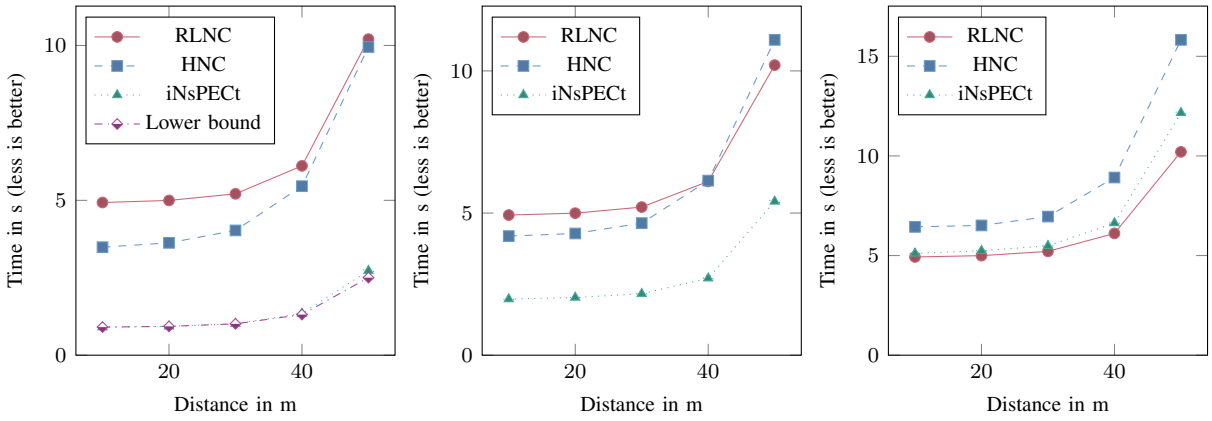
not have full rank. Each simulation is run for 200 s simulated time and executed 5 times. Each repeated simulation uses a separate sub-stream of ns-3's MRG32k3a pseudo-random number generator to ensure uncorrelated results [26]. Pseudo-randomness is used in the simulation's wireless fading model, the ns-3 bit error model, which is affected by fading and path loss, generation of NC coefficients, and exponentially distributed variations in packet send times that we use to avoid collisions and other synchronization effects. Since generations take much less than 200 s to transmit, hundreds of transmitted generations contribute to a statistically meaningful sample size. All figures in this section show the sample mean and *95% confidence intervals* (assuming normal distribution). Error bars may not be visible when the confidence is very high.

B. Small-scale topology

Figure 3a shows the small-scale topology: one source S and four nodes N_1 to N_4 are arranged in a partial grid with 30 m grid width. Due to the grid layout, nodes N_1 and N_2 have 30 m distance from the center-positioned source, whereas nodes N_2 and N_4 are 42 m away, which results in lower packet delivery probability (PDR) for these nodes. We evaluate a small generation ($n = 10$) with small layers $\boldsymbol{r} = (2, 2, 3, 3)$ to highlight the effects of the layer selection and feedback rates.

The mean time until all sink nodes are able to decode layers 1 to 4, respectively, is given in Figure 4 for two different feedback rates. Results for frequent feedback are shown in Figure 4a: the y-axis shows the average time from beginning to transmit the current generation until a layer can be decoded by a node. The x-axis gives the individual layers, which are inherently cumulative due to the hierarchical nature of layers. It can be seen that using RLNC, the time until all layers can be decoded is identical for all layers. Since RLNC offers maximum protection against erasures and has the lowest chance to send linearly dependent combinations, it gives us the optimal decoding time for layer 4 (and thus all layers) in the last column. The HNC strategy, being fully randomized and independent of any feedback, provides faster recovery of the highest prioritized first layer, identical decoding time for the second layer, and significantly worse delay than RLNC for the third and fourth layer. The proposed algorithm, iNsPECt, consistently outperforms HNC between 35.9 % and 72.3 %. Also, the delay is just 4.5 % higher than the *lower bound* for the most highly prioritized 1st layer (compared to 277.5 % for HNC). The layer 4 decoding delay of iNsPECt is only 17.3 % higher than the optimal RLNC delay. These 17.3 % analogously give the overhead imposed by the prioritization scheme, since the additional delay corresponds to the number of linearly dependent combinations that are due to prioritization. In comparison, this overhead is 83.1 % for HNC.

In a second step, we lowered the feedback rate to $1/3 \cdot \lambda_{\text{data}}$, which is quite low compared to the individual layer sizes: without losses, another layer has to be selected every two or three linear combinations or all subsequently sent combinations are linearly dependent. The results are shown in Figure 4b in a format identical to Figure 4a: RLNC and HNC, working without feedback, are unaffected by the change. iNsPECt is still faster than HNC for all layers, but due to the lack of recent feedback and the resulting uncertainty about the neighbors' decoder states, the algorithm resorts to sending layers of lower priority than necessary, which have a much lower chance of



(a) Layer 1 (b) Layers 1–2 (c) All layers (1–4)

Fig. 5. *Larger* topology results: per-layer delay for different (cumulative) layers and varying distances.

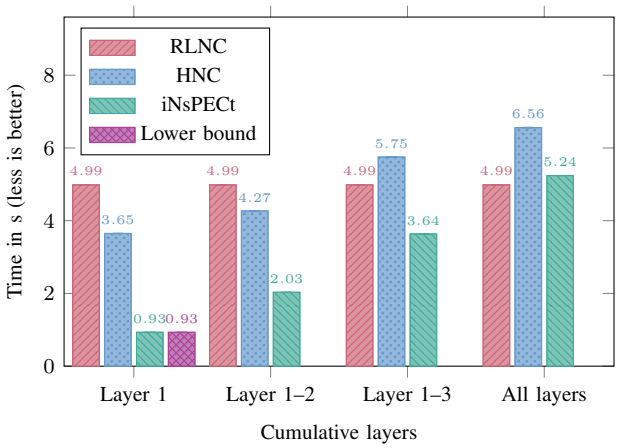


Fig. 6. Random topology results: average per-layer delay.

linear dependency. The downside of this approach can best be seen in the second and third layers, where decoding delay is still 38.8 % and 33.8 % better than HNC, but also increases by 28.6 % and 13.1 % compared to the better feedback rate scenario. A positive aspect of resorting to lower priority linear combinations is that linear dependency is less likely, which can be seen best at the fourth layer, where decoding time is not significantly different from the former scenario. This means that albeit iNsPECt's per-layer delay is not as low as before, it is still better than HNC *and* total prioritization overhead compared to RLNC does not increase at all.

C. Larger topology and random topology

We now demonstrate scalability to larger ad-hoc networks with multi-hop requirements. The larger network topology is shown in Figure 3b and has 16 nodes in total: 4 source nodes and 12 non-source nodes. As before, nodes are arranged in a grid, but now we have four nodes in each row. We simulate a larger generation with $n = 50$ and 4 layers with layer sizes $\boldsymbol{r} = (10, 10, 15, 15)$. Figure 5 shows the simulation results for medium feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/2$) and varying grid distances x between neighbor nodes from 10 m to 50 m. Figures 5a to 5c give decoding delay for layer 1, layers 1–2, and layers 1–4, respectively.

iNsPECt allows nodes to retrieve the most highly prioritized 1st layer 74.1 %, 74.7 %, and 72.6 % faster than HNC for node distances of 10 m, 30 m, and 50 m, respectively. Also, the first layer's retrieval time with iNsPECt almost matches the lower bound; only at 40 m and 50 m distance, the results show 2.4 % and 9.0 % delay for the 1st layer.

Results look similar for the second layer in Figure 5b, where iNsPECt yields a 52.9 % to 51.2 % reduced per-layer delay over HNC. Interestingly, the benefit of HNC over RLNC for prioritized layers diminishes with greater distances between nodes (and thus lower PDR): at 40 m distance between nodes, HNC gives roughly the same per-layer delay as RLNC. We attribute the poor performance of HNC with low PDR in the large-scale scenario to inefficient multi-hop capabilities: when the source has few opportunities to successfully transmit linear combinations to the inner nodes in the network, low priority linear combinations are more useful, because they have much higher chance of being innovative.

As expected, Figure 5c shows that the last layer – and thus all layers due to the hierarchical layer structure – is decoded the fastest with RLNC, which has the highest level of error protection against packet loss and the lowest chance of sending linear combinations of layers that already have full rank in neighbors. The delay given in Figure 5c is a direct indicator for the total number of transmissions required for sending one full generation. Therefore, it is also indicative for achievable throughput. For distances at or below 30 m, iNsPECt has at most 5.4 % overhead compared the optimum RLNC. This overhead increases to 8.5 % at 40 m distance between neighbors and 19.2 % at 50 m. HNC, in comparison, results in a message overhead of 55.1 % over RLNC.

Last, we verify the system's properties in a set of larger randomized topologies. Figure 6 shows mean per-layer delay for twenty different topologies where the nodes' locations are selected uniform at random in a $90\,\text{m} \times 90\,\text{m}$ square. Again,

we see a better performance of iNsPECt than HNC for all layers and a low total overhead of only 5.1 % compared to RLNC. Notably, iNsPECt enables decoding the most highly prioritized first layer 74.5 % faster than HNC.

D. Summary

We evaluated iNsPECt in both smaller and larger scenarios and compared it to HNC and RLNC for different feedback rates, distances, and topologies. iNsPECt significantly outperforms HNC in every scenario that we tested. Having only sporadic feedback and thus outdated decoder state information has no effect on the system's overhead in terms of linear dependency, but it increases decoding delay for some layers, albeit keeping delay significantly lower than HNC. Remarkably, in all scenarios, that is, for small and large distances, for high and low feedback rates, and for the small and larger-scale scenarios, the most highly prioritized layer's decoding delay was within 10 % of the optimum. The overhead of iNsPECt compared to RLNC over all scenarios is consistently less than 20 %, which we consider a low cost for having prioritization.

VII. Conclusion and Future Work

We describe a protocol that addresses a principal problem of existing prioritized network coding protocols. Namely, we answer the question which layer to use for generating linear combinations. Towards this end, we propose a novel, distributed algorithm, iNsPECt, that leverages limited feedback containing each layer's subspace dimension. iNsPECt defines two performance indicators that enable it to deviate from a greedy strategy in order to reduce prioritization overhead without affecting prioritization performance. Our evaluation shows that the proposed algorithm consistently outperforms HNC and, under good network conditions, approaches the lower bound on required transmissions that is achieved by non-prioritized RLNC. In addition, the highest priority layer's decoding delay is nearly optimal in all scenarios. Our results demonstrate that (1) prioritized network coding can be realized with low overhead, and (2) that even small feedback messages are sufficient for effective prioritization in such systems.

A future research direction is to observe the algorithm's performance in a more sophisticated network protocol: it is conceivable that observed channel conditions and limited network topology information supplied by such a protocol could be utilized to better estimate neighbors' decoder state if recent feedback is unavailable. In particular, we would expect a reduced per-layer delay for mid-priority layers when expected packet-loss rates are incorporated into the layer selection process that is used for generating linear combinations.

Acknowledgments

We thank Sebastian Henningsen for helpful discussion. Also, we thank our anonymous reviewers for their constructive feedback.

References

[1] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: An instant primer," 2006.

[2] R. Ahlswede, N. Cai, S. Y. R. Li, *et al.*, "Network information flow," Jul. 2000.

[3] Baochun Li and Di Niu, "Random Network Coding in Peer-to-Peer Networks: From Theory to Practice," 2011.

[4] Christos Gkantsidis, John Miller, and Pablo Rodriguez, "Comprehensive view of a live network coding P2P system," presented at the Internet Measurement Conference, 2006.

[5] X. Chu and Y. Jiang, "Random linear network coding for peer-to-peer applications," Jul. 2010.

[6] E. Magli, M. Wang, P. Frossard, *et al.*, "Network coding meets multimedia: A review," 2013.

[7] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007.

[8] S.-Y. Li, Q. Sun, and Z. Shao, "Linear network coding: Theory and algorithms," Mar. 2011.

[9] T. Ho, R. Koetter, M. Medard, *et al.*, "The benefits of coding over routing in a randomized setting," 2003.

[10] O. Trullols-Cruces, J. M. Barcelo-Ordinas, and M. Fiore, "Exact Decoding Probability Under Random Linear Network Coding," Jan. 2011.

[11] K. Nguyen, T. Nguyen, and S. c Cheung, "Peer-to-peer streaming with hierarchical network coding," in *2007 IEEE International Conference on Multimedia and Expo*, Jul. 2007.

[12] D. Vukobratović and V. Stanković, "Unequal error protection random linear coding for multimedia communications," in *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop On*, IEEE, 2010.

[13] R. Naumann, S. Dietzel, and B. Scheuermann, "Best of both worlds: Prioritizing network coding without increased space complexity," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, Nov. 2016.

[14] M. Esmaeilzadeh, P. Sadeghi, and N. Aboutorab, "Random Linear Network Coding for Wireless Layered Video Broadcast: General Design Methods for Adaptive Feedback-Free Transmission," Feb. 2017.

[15] Z. Yan, H. Xie, and B. W. Suter, "Rank deficient decoding of linear network coding," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013.

[16] Shenglan Huang, Michele Sanna, Ebroul Izquierdo, *et al.*, "Optimized scalable video transmission over P2P network with hierarchical network coding," presented at the ICIP, 2014.

[17] P. Chau, S. Kim, Y. Lee, *et al.*, "Hierarchical random linear network coding for multicast scalable video streaming," in *Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA)*, IEEE, 2014.

[18] B. Shrader and N. M. Jones, "Systematic wireless network coding," in *MILCOM 2009 - 2009 IEEE Military Communications Conference*, Oct. 2009.

[19] J. Claridge and I. Chatzigeorgiou, "Probability of Partially Decoding Network-Coded Messages," 2017.

[20] Y. Wu, P. Chou, K. Jain, *et al.*, "A comparison of network coding and tree packing," in *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium On*, IEEE, 2004.

[21] S. Chachulski, M. Jennings, S. Katti, *et al.*, "MORE: A network coding approach to opportunistic routing," 2006.

[22] T. R. Henderson, M. Lacage, G. F. Riley, *et al.*, "Network simulations with the ns-3 simulator," 2008.

[23] M. Lacage and T. R. Henderson, "Yet another network simulator," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, ACM, 2006.

[24] H. Hashemi, "The indoor radio propagation channel," 1993.

[25] S. Phaiboon, "Space Diversity Path Loss in a Modern Factory at frequency of 2.4 GHz," 2014.

[26] P. L'Ecuyer, R. Simard, E. J. Chen, *et al.*, "An Object-Oriented Random-Number Package with Many Long Streams and Substreams," Dec. 2002.

A Blockchain Consensus Protocol With Horizontal Scalability

Kelong Cong
École Polytechnique Fédérale de Lausanne
Email: kelong.cong@epfl.ch

Zhijie Ren
Delft University of Technology
Email: z.ren@tudelft.nl

Johan Pouwelse
Delft University of Technology
Email: peer2peer@gmail.com

***Abstract*—Blockchain technology has the potential to decentralise many traditionally centralised systems. However, scalability remains a key challenge. A horizontally scalable solution, where performance increases by adding more nodes, would move blockchain systems one step closer to ubiquitous use. We design a novel blockchain system called CHECO. Each node in our system maintains a personal hash chain, which only stores transactions that the node is involved in. A consensus is reached on special blocks called checkpoint blocks rather than on all transactions. Checkpoint blocks are effectively a hash pointer to the personal hash chains; thus a single checkpoint block may represent an arbitrarily large set of transactions. We introduce a validation protocol so that any node can check the validity of any transaction. Since transaction and validation protocols are point-to-point, we achieve horizontal scalability. We analytically evaluate our system and show a number of highly desirable correctness properties such as consensus on the validity of transactions. Further, we give a free and open-source implementation of CHECO and evaluate it experimentally. Our results show a strong indication of horizontal scalability.**

I. INTRODUCTION

The first blockchain system—Bitcoin—is almost ten years old. Its market capitalisation is nearly $200 billion USD at the time of writing [1]. We can be reasonably sure that such systems, even if their application is still somewhat limited, are here to stay in the foreseeable future. Driven by the success of Bitcoin, we see a renaissance of consensus research [2]–[4], where the primary focus is to improve the scalability of blockchain systems, which is due to the inefficiencies of the consensus mechanism—proof-of-work (PoW). For example, Bitcoin can only do 7 transactions per second (TPS) at most [5]. While adjusting the block size (which Bitcoin has recently done via SegWit [6]) and/or the block interval may increase TPS, it also leads to centralisation as larger blocks take longer to propagate through the network, putting miners that do not have a fast network at a disadvantage [7]. Furthermore, due to the bandwidth and latency of today's network, it is not possible to achieve more than 27 TPS from simply adjusting the block size or block interval [7].

Related work. Many approaches exist for improving the scalability of early blockchain systems. Off-chain transactions make use of the fact that if nodes make frequent transactions, then it is not necessary to store every transaction on the blockchain, only the net settlement is needed. The best examples are Lightning Network [8] and Duplex Micropayment Channels [9]. It promises significant scalability improvements, but complicates user experience and leads to centralisation. That is, each node must deposit a suitable amount of Bitcoins into a multi-signature account. A low deposit would not allow large transactions. A high deposit locks the user from using much of their Bitcoins outside the channel. In addition, the user must proactively check whether the counterparty has broadcasted an old channel state so that the user does not lose Bitcoins. Moreover, creating channels with sufficient balance and also keeping it online to act as a router is expensive. A casual user is not capable of such tasks, leading to centralisation.

Another way to improve transaction rate is to use traditional Byzantine consensus algorithms such as PBFT [10] in a permissioned ledger such as Hyperledger Fabric [11]. In essence, such systems contain a fixed set of nodes, called validating peers, that run a Byzantine consensus algorithm to decide on new blocks. They can achieve much higher transaction rates, e.g., 10,000 TPS if the number of validating peers is under 16 for PBFT [12, Section 5.2]. However, these systems do not scale, e.g., the transaction rate drops to under 5000 TPS when the number of validating peer is 64 [12, Section 5.2]. Moreover, the validating peers are predetermined which makes the system unsuitable for the open internet.

Recent research has developed a class of hybrid systems which uses PoW for committee election, and Byzantine consensus algorithms to agree on transactions, e.g., ByzCoin [3] and Solidus [13]. This design is primarily for permissionless systems because the PoW leader election aspect prevents the Sybil attack [14]. It overcomes the early blockchain scalability issue by delegating the transaction validation to a Byzantine consensus protocol. A tradeoff of such systems is that they cannot guarantee a high level of fault tolerance when there is a large number of malicious nodes (but less than a majority). ByzCoin and Solidus all have some probability of electing more than t Byzantine nodes into the committee, where t is typically just under a third of the committee size (a lower bound of Byzantine consensus [15]). Again, because these systems must reach consensus on all transactions, none of them achieves horizontal scalability.

Finally, a technique that does achieve horizontal scalability is sharding, e.g., Elastico [2] and OmniLedger [4]. It involves grouping nodes into multiple committees of constant size, also known as shards, and nodes within a single shard run a

ISBN 978-3-903176-08-9

Byzantine consensus algorithm to agree on a set of transactions that belong to that specific shard. The number of shards grows linearly with respect to the total computational power of the network; hence the transaction rate also grows linearly. The limitation of sharding is that it is only optimal if transactions stay in the same shard. In fact, Elastico cannot atomically process inter-shard transactions. OmniLedger has an inter-shard transaction protocol but choosing a shard size that matches the transaction characteristics of the network is difficult. An inadequate shard size would result in a large number of inter-shard transactions which would hinder scalability.

Research question. Thus far, there are no systems that achieve horizontal scalability in the general case, which leads to the goal of this work. Hence, the research question which we wish to answer is as follows.

> *How can we design a horizontally scalable blockchain consensus protocol?*

Concretely, a blockchain consensus protocol should be application neutral. For example, PoW is application neutral because transaction semantics does not affect it, i.e. it can be applied in different applications such as cryptocurrency (Bitcoin) and domain name system (Namecoin [16]). Further, we are interested in horizontal scalability in the general case as it enables ubiquitous use. That is, adding more nodes to the network should result in higher transaction throughput.

Contribution. The key insight is not to reach consensus using an existing consensus algorithm on transactions themselves, but on special blocks called checkpoint blocks, such that transactions are nevertheless verifiable at a later stage by any node in the network. Our main contributions are the following.

- We formally introduce a blockchain system—CHECO[1]. It uses individual hash chains and checkpoints on every node to achieve horizontal scalability in the general case for the first time.
- We analyse CHECO to ensure correctness according to our definition.
- We provide an implementation and then experiment with up to 1200 nodes, our results show strong evidence of horizontal scalability.

Roadmap. In Section II, we give the problem description and our system model. Section III gives the formal system architecture. In Section IV, we discuss a few design variations and their tradeoffs. We argue the correctness and fault tolerance properties of our system in Section V. Then we evaluate our system experimentally in Section VI. Finally, we conclude our work in Section VII.

II. Problem Description

We introduce the problem as a modified Byzantine consensus problem. The modification is primarily derived from the need of horizontal scalability, which is not a part of a typical Byzantine consensus problem. In our model, we consider N nodes, t of which are Byzantine. Nodes in our system make transactions with each other. Transactions can be in one of three states—*valid*, *invalid* and *unknown*. We seek a protocol that satisfies the following properties.

- *Agreement*: If any correct node decides on the validity of a transaction, except when it is *unknown*, then all other correct nodes are able to reach the same conclusion or decide *unknown*.
- *Validity*: If a transaction is valid, then it must have been created by two honest nodes.
- *Scalability*: If every node makes transactions at the same rate, then as N increases, the global transaction rate should increase linearly w.r.t. N.

Note that the agreement property is similar, but a relaxed version of what is often seen in a Byzantine consensus problem. Namely, the property only holds if honest nodes do not output *unknown*. For example, for a transaction, it is fine if two honest nodes output *valid* and *unknown*, but they should never output *valid* and *invalid*. Our problem does not have a termination property. Instead, nodes are incentivised to complete the protocol execution otherwise they risk economical loss; we describe this phenomenon in Section V-B.

The problem is purposefully made to be application neutral, i.e. there are no constraints on the semantics of transactions. This formulation is so that the protocol can act as a building block to many applications. Thus, we do not consider global fork prevention or detection, as some application may not need such strong guarantees such as the accounting of internet traffic in Tribler [17], [18]. On the other hand, we give two alternative constructions that do perform fork detection in Section IV-C.

System model. We assume purely asynchronous channels with eventual delivery. Thus, in no stage of the protocol are we allowed to make timing assumptions. The adversary has full control of the delivery schedule and the message ordering of all messages.

Security assumptions. The malicious nodes are Byzantine, meaning that there are no restrictions on the type of failure. We use a static, round-adaptive corruption model. That is, if a round has started, the corrupted nodes cannot change until the next round. We assume there exists a Public Key Infrastructure (PKI), and nodes are identified by their unique and permanent public key. This assumption implies that we work in the permissioned model. Finally, we use the random oracle (RO) model, i.e. calls to the random oracle are denoted by $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, where $\{0,1\}^*$ denotes the space of finite binary strings and λ is the security parameter. Under the RO model, the probability of successfully computing the inverse of the hash function is negligible with respect to λ [19].

III. System Architecture

To describe CHECO, we first give an informal overview and then move on to the formal description.

Early blockchain systems that use a global ledger are difficult to scale because every node must reach consensus on all the transactions that ever existed. Instead, we introduce an alternative architecture where every node has their own genesis

[1]Derived from "CHEckpoint COnsensus".

block and hash chain. The nodes only store transactions (TX) that they are involved in on their hash chains. Transactions are stored in TX blocks, and every block only contains one transaction. A transaction between two nodes should, therefore, result in two TX blocks on their respective hash chains. We introduce a special block called checkpoint (CP) block, which represents the state of a hash chain in the form of a hash pointer. Then, a collection of CP blocks from all nodes would represent the state of the whole system. A visualisation can be seen in Figure 1.

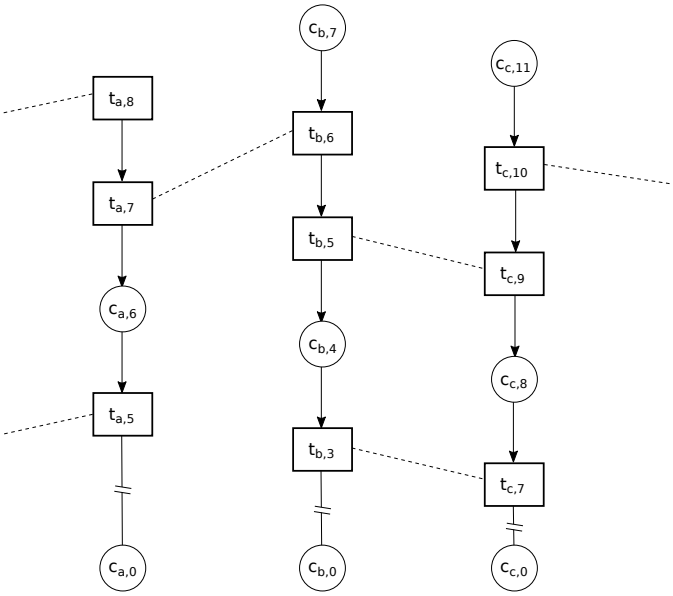


Fig. 1: Visualisation of the data structure used in CHECO. $t_{u,i}$ represents a TX block on u's chain with a sequence number i. $c_{v,j}$ represents a CP block on v's chain with a sequence number j. The blocks at the ends of the dotted lines are pairs of each other. Blocks of sequence number 0 (e.g., $c_{c,0}$) are genesis blocks.

CHECO consists of three protocols—consensus protocol, transaction protocol and validation protocol—all interacting with the distributed hash chain data structure described above. The primary protocol is the consensus protocol, which can be seen as a technique of running infinitely many times of an existing Byzantine consensus algorithm (in this work we use the asynchronous common subset protocol described in HoneyBadgerBFT [12]), starting a new execution immediately after the previous one is completed. Nodes create new CP blocks at the end of every execution. This approach is necessary because blockchain systems always need to reach consensus on new values proposed by the nodes in the system, or CP blocks in our case.

The communication complexity of Byzantine consensus algorithms typically grows polynomially w.r.t the number of nodes, which prohibits us from running it on a large network. Thus, at the beginning of every Byzantine consensus algorithm execution, we randomly elect a set of nodes—called facilitators—to collect CP blocks from every other node and use those blocks as the input to the Byzantine consensus algorithm. After the algorithm completes, the facilitators output a set of CP blocks which we call the consensus result, which is then propagated to the network. Using the result, nodes are allowed to create new CP blocks, and then the next algorithm execution begins.

The transaction protocol is a simple request and response protocol. The nodes exchange one round of messages and create new TX blocks on their respective chains. Thus, as we mentioned before, one transaction should result in two TX blocks.

The consensus and transaction protocol by themselves do not provide a mechanism to detect malicious behaviour such as tampering. Thus, we need a validation protocol to counteract such behaviour. When a node wishes to validate one of its transactions, it asks the counterparty for the *agreed fragment* of the transaction. Which is a section of the counterparty's chain beginning and ending with CP blocks but contains the TX block belonging to that transaction, where the CP blocks must be in consensus. Upon the counterparty's response, the node checks whether the CP blocks are, in fact, in some consensus result and among other conditions. The transaction is valid if these conditions are satisfied. Since the transaction and validation protocols only make point-to-point communication, we achieve horizontal scalability.

The following sections give the formal description.

A. CHECO *data structure*

Each node u has a public and private key pair—pk_u and sk_u, and a hash chain B_u. The chain consist of blocks $B_u = \{b_{u,i} : i \in \{0, \ldots, h-1\}\}$, where $b_{u,i}$ is the ith block of u, and h is the height of the block (i.e. $h = |B_u|$). We use $b_{u,h-1}$ to denote the latest block. There are two types of blocks, TX blocks and CP blocks. If T_u is the set of all TX blocks in B_u and C_u is the set of all CP blocks is B_u, then $T_u \cup C_u = B_u$ and $T_u \cap C_u = \varnothing$. The notation $b_{u,i}$ is generic over the block type.

Definition 1 (Transaction block)**.** *The TX block is a six-tuple, i.e*

$$t_{u,i} = \langle H(b_{u,i-1}), seq_u, txid, pk_v, m, sig_u \rangle.$$

We describe each item in turn.

1) $H(b_{u,i-1})$ *is the hash pointer to the previous block.*
2) seq_u *is the sequence number which should equal* i*.*
3) $txid$ *is the transaction identifier, it should be generated using a cryptographically secure pseudo-random number generator by the initiator of the transaction.*
4) pk_v *is the public key of the counterparty* v*.*
5) m *is the transaction message, which can be seen as an arbitrary string.*
6) sig_u *is the signature created using* sk_u *on the concatenation of the binary representation of the five items above.*

TX blocks come in pairs. In particular, for every block

$$t_{u,i} = \langle H(b_{u,i-1}), seq_u, txid, pk_v, m, sig_u \rangle$$

there exists one and only one pair

$$t_{v,j} = \langle H(b_{v,j-1}), seq_v, txid, pk_u, m, sig_v \rangle,$$

if the nodes follow the transaction protocol (described in Section III-C). Note that the $txid$ and m are the same, and the public keys refer to each other. Thus, given a TX block, these properties allow us to identify its pair.

Definition 2 (Checkpoint block and genesis block)**.** *The CP block is a five-tuple, i.e.*

$$c_{u,i} = \langle H(b_{u,i-1}), seq_u, H(\mathcal{C}_r), r, sig_u \rangle,$$

where $\mathcal{C}_r$ is the consensus result (which we describe next in Definition 3) in round r, the other items are the same as the TX block definition.

The genesis block in the chain must be a CP block in the form of

$$c_{u,0} = \langle H(\epsilon), 0, H(\epsilon), 0, sig_u \rangle,$$

where ϵ is the empty string. The genesis block is unique because every node has a unique public and private key pair.

Definition 3 (Consensus result)**.** *Our consensus protocol runs in rounds, where the first round is defined to be 1 and it is incremented after every execution of the consensus protocol. The consensus result, output of the consensus protocol, is a tuple, i.e.*

$$\mathcal{C}_r = \langle r, C \rangle,$$

where C is a set of CP blocks agreed by the facilitators of round r.

Next we define a property which results from the interleaving nature of CP and TX blocks. It is used in our validation protocol (discussed in Section III-D).

Definition 4 (Enclosure and agreed enclosure)**.** *If there exists a tuple $\langle c_{u,a}, c_{u,b} \rangle$ for a TX block $t_{u,i}$, where*

- *$c_{u,a}$ is the closest CP block to $t_{u,i}$ with a lower sequence number and*
- *$c_{u,b}$ is the closest CP block to $t_{u,i}$ with a higher sequence number,*

then $\langle c_{u,a}, c_{u,b} \rangle$ is the enclosure of $t_{u,i}$. Some TX blocks may not have any enclosure, then their enclosure is $\perp$. Agreed enclosure is the same as enclosure with an extra constraint where the CP blocks must be in some consensus result $\mathcal{C}_r$.

Definition 5 (Fragment and agreed fragment)**.** *If the enclosure of some TX block $t_{u,i}$ is $\langle c_{u,a}, c_{u,b} \rangle$, then its fragment $F_{u,i}$ is defined as $\{b_{u,i} : a \leq i \leq b\}$. Similarly, agreed fragment has the same definition as fragment but using agreed enclosure. For convenience, the function agreed_fragment$(t_{u,i})$ outputs the agreed fragment of $t_{u,i}$ if it exists, otherwise $\perp$.*

B. Consensus Protocol

Our scalable consensus protocol Π_c uses an asynchronous common subset (ACS) protocol as the key building block. The objectives of the protocol are to allow honest nodes always make progress (in the form of creating new CP blocks), compute correct consensus result in every round and have an unbiased election of facilitators. We formally define the desired properties below.

Definition 6 (CHECO consensus protocol)**.** *A CHECO consensus protocol is correct if the following holds for every round r.*

1) Agreement*: If one correct node outputs a set of facilitators $\mathcal{F}_r$, then every node outputs $\mathcal{F}_r$*
2) Validity*: If any correct node outputs $\mathcal{F}_r$, then*
   a) *$|\mathcal{C}_r| \geq N - t^2$, and*
   b) *$|\mathcal{F}_r| = n$.*
3) Termination*: Every correct node eventually outputs some $\mathcal{F}_r$.*

1) Bootstrap Phase: To bootstrap, imagine that there is some bootstrap oracle that initiates the correct program on every node, meaning that it satisfied the properties in Definition 6. In practice, the bootstrap oracle is most likely a group of software developers (representing different organisations) that agreed to work together to set up the system and assign the facilitators of round 1. The number of facilitators is n, we discuss the trade-offs for different values of n in Section VI. This concludes the bootstrap phase. For any future rounds, the consensus phase is used.

2) Consensus Phase: For any node u, the consensus phase begins when $\mathcal{F}_r$ is available and the latest block is $c_{u,h-1}$. Note that $\mathcal{F}_r$ indicates the facilitators that were elected using results of round r and are responsible for driving the ACS algorithm in round $r+1$. The goal is to reach agreement on a set of new facilitators $\mathcal{F}_{r+1}$ that satisfies the four properties in Definition 6.

There are two scenarios in the consensus phase. First, if u is not the facilitator, it sends $\langle \texttt{cp\_msg}, c_{u,h-1} \rangle$ to all the facilitators. Second, if u is a facilitator, it waits for $N - t$ messages of type `cp_msg`. Invalid messages are removed, which are blocks with invalid signatures and blocks signed by the same key. With a sufficient number of `cp_msg` messages, it begins the ACS algorithm and some $\mathcal{C}'_{r+1}$ should be agreed upon by the end of it. Duplicates and blocks with invalid signatures are again removed from $\mathcal{C}'_{r+1}$ and we call the final result $\mathcal{C}_{r+1}$. We have to remove invalid blocks a second time because the adversary may send different CP blocks to different facilitators, which results in invalid blocks in the ACS output, but not in any of the inputs.

The core of the consensus phase is the ACS algorithm, which is described in HoneyBadgerBFT [12]. We do not use the full HoneyBadgerBFT due to the following. First, the transactions in HoneyBadgerBFT are first queued in a buffer and the main consensus algorithm starts only when the buffer reaches an optimal size. We do not have an infinite stream of CP blocks, thus buffering is unsuitable. Second, HoneyBadgerBFT uses threshold encryption to hide the content of the transactions. But we do not reach consensus on transactions,

[2] $\mathcal{C}_r$ is a tuple but we abuse the notation here by writing $|\mathcal{C}_r|$ to mean the number of CP blocks in the second element of $\mathcal{C}_r$.

only CP blocks; the content of the CP blocks are not sensitive so there is no need to hide it.

When $\mathcal{F}_r$ finish the ACS execution and reach agreement on $\mathcal{C}_{r+1}$, they immediately broadcast two messages to all the nodes—first the consensus message $\langle \texttt{cons\_msg}, \mathcal{C}_{r+1} \rangle$, and second the signature message $\langle \texttt{cons\_sig}, r, sig \rangle$. The reason for sending `cons_sig` is the following. The channels are not authenticated, and there are no signatures in $\mathcal{C}_{r+1}$. If a non-facilitator sees some $\mathcal{C}_{r+1}$, it cannot immediately trust it because it may have been forged. Thus, to guarantee authenticity, every facilitator sends an additional message that is the signature of $\mathcal{C}_{r+1}$.

Upon receiving $\mathcal{C}_{r+1}$ and at least $n - t$ valid signatures, u performs two tasks. First, it creates a new CP block using $\mathsf{new\_cp}(\mathcal{C}_{r+1})$, described in Algorithm 1. Second, it computes the new facilitators using $\mathsf{get\_facilitator}(\mathcal{C}_{r+1}, n)$, described in Algorithm 2, and updates its facilitator set to the result. This concludes the consensus phase and brings us back to the state at the beginning of the consensus phase, so a new round can be started.

Our protocol has some similarities with synchronizers [20, Chapter 10] because it is effectively a technique to introduce synchrony in an asynchronous environment. If we consider the facilitators as a collective authority, then it can be seen as a synchronizer that sends pulse messages (in the form of `cons_msg` and `cons_sig`) to indicate the start of a new clock pulse. Every node then sends a completion messages (in the form of `cp_msg`) to the new collective authority to indicate that they are ready for the next pulse.

Algorithm 1 Function $\mathsf{new\_cp}(\mathcal{C}_r)$ runs in the context of the caller u. It creates a new CP block and appends it to u's chain.

$\langle r, \_ \rangle \leftarrow \mathcal{C}_r$
$h \leftarrow |B_u|$
$c_{u,h} \leftarrow \langle \mathsf{H}(b_{u,h-1}), h, \mathsf{H}(\mathcal{C}_r), r, sig_u \rangle$
$B_u \leftarrow B_u \cup c_{u,h}$

Algorithm 2 Function $\mathsf{get\_facilitator}(\mathcal{C}_r, n)$ takes the consensus result $\mathcal{C}_r$ and an integer n, then sorts the CP blocks C by the luck value (the λ-expression), and outputs the smallest n elements.

$\langle r, C \rangle \leftarrow \mathcal{C}_r$
return $\mathsf{take}(n, \mathsf{sort\_by}(\lambda x.\mathsf{H}(\mathcal{C}_r || pk \text{ of } x), C))$

C. Transaction Protocol

The TX protocol Π_t, shown in Algorithm 4, is run by all nodes. Nodes that wish to initiate a transaction calls $\mathsf{new\_tx}(pk_v, m, txid)$, described in Algorithm 3, with the intended counterparty v identified by pk_v and message m. $txid$ should be a uniformly distributed random value, i.e. $txid \in_R \{0,1\}^{256}$. Then the initiator sends $\langle \texttt{tx\_req}, t_{u,h} \rangle$ to v.

A key feature of Π_t is that it is non-blocking. At no time in Algorithm 3 or Algorithm 4 do we need to hold the chain state and wait for some message to be delivered before committing a new block to the chain. This allows for a high level of concurrency where we can call many $\mathsf{new\_tx}(\cdot)$ and send multiple `tx_req` messages simultaneously without waiting for the corresponding `tx_resp` messages.

Algorithm 3 Function $\mathsf{new\_tx}(pk_v, m, txid)$ generates a new TX block and appends it to the caller u's chain. It is executed in the private context of u, i.e. it has access to the sk_u and B_u.

$h \leftarrow |B_u|$
$t_{u,h} \leftarrow \langle \mathsf{H}(b_{u,h-1}), h, txid, pk_v, m, sig_u \rangle$
$B_u \leftarrow B_u \cup \{t_{u,h}\}$

Algorithm 4 Π_t runs in the context of node u.

Upon $\langle \texttt{tx\_req}, t_{v,j} \rangle$ from v
  $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$
  $\mathsf{new\_tx}(pk_u, m, txid)$
  store $t_{v,j}$ as the pair of $t_{u,h}$
  send $\langle \texttt{tx\_resp}, t_{u,h} \rangle$ to v
Upon $\langle \texttt{tx\_resp}, t_{v,j} \rangle$ from v
  $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$
  store $t_{v,j}$ as the pair of the TX with identifier $txid$

D. Validation Protocol

Up to this point, we do not provide a mechanism to detect tampering. The validation protocol Π_v aims to solve this issue. The protocol is also a request-response protocol. Before explaining the protocol itself, we first define what it means for a transaction to be valid.

1) Validity Definition: A transaction can be in one of three states in terms of validity—*valid*, *invalid* and *unknown*. Given a fragment $F_{v,j}$, the validity of the TX block $t_{u,i}$ with its corresponding fragment $F_{u,i}$ is captured by the function $\mathsf{get\_validity}(t_{u,i}, F_{u,i}, F_{v,j})$ in Algorithm 5. Note that $t_{u,i}$ and $F_{u,i}$ are assumed to be valid, otherwise the node calling the function would have no point of reference. This is not difficult to achieve because typically the caller is u, so it knows its own TX block and the corresponding agreed fragment. If the caller is not u, it can always query for the agreed fragment that contains the transaction of interest from u.

We stress that the *unknown* state means that the verifier does not have enough information to make progress in Π_v. If enough information is available at a later time, then the verifier will output either *valid* or *invalid*.

Note that the validity is on a transaction, i.e. two TX blocks that form a pair. It is defined this way because the malicious sender may create new TX blocks in their own chain but never send `tx_req` messages. In that case, it may seem that the counterparty, who is honest, purposefully omitted TX blocks. But in reality, it was the malicious sender who did not follow the protocol. Thus, in such cases, the whole transaction identified by its $txid$ is marked as invalid.

Algorithm 5 Function get_validity$(t_{u,i}, F_{u,i}, F_{v,j})$ validates the transaction represented by $t_{u,i}$. We assume $F_{u,i}$ is always correct and contains $t_{u,i}$. $F_{v,j}$ is the corresponding fragment received from v.

if $F_{v,j}$ is not a fragment created in the same round as $F_{u,i}$ **then**
  return *unknown*
$\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{u,i}$
if number of blocks of $txid$ in $F_{v,j} \neq 1$ **then**
  return *invalid*
$t_{v,j} \leftarrow$ the TX block with $txid$ in $F_{v,j}$
$\langle \_, \_, txid', pk_u, m', \_ \rangle \leftarrow t_{v,j}$
if $m \neq m' \vee txid \neq txid'$ **then**
  return *invalid*
if $t_{u,i}$ is not signed by $pk_u \vee t_{v,j}$ is not signed by pk_v **then**
  return *invalid*
return *valid*

2) Validation Protocol: Our validation protocol Π_v, shown in Algorithm 6, is designed to classify transactions according to the aforementioned validity definition. If u wishes to validate some TX with ID $txid$ and counterparty v, it sends $\langle \texttt{vd\_req}, txid \rangle$ to v. The desired properties are as follows.

Definition 7 (CHECO validation protocol)**.** *A* CHECO *validation protocol is correct if the following properties hold.*

1) Agreement*: If any correct node decides on the validity of a transaction, except when it is* unknown*, then all other correct nodes are able to reach the same conclusion or decide* unknown*.*
2) Validity*: The validation protocol outputs the correct result according to the validity definition above.*
3) Liveness*: Any valid (invalid) transaction is marked as valid (invalid) eventually.*

Algorithm 6 Π_v which runs in the context of u

Upon $\langle \texttt{vd\_req}, txid \rangle$ from v
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  $F_{u,i} \leftarrow$ agreed_fragment$(t_{u,i})$
  send $\langle \texttt{vd\_resp}, txid, F_{u,i} \rangle$ to v
Upon $\langle \texttt{vd\_resp}, txid, F_{v,j} \rangle$ from v
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  if $F_{u,i}$ and $F_{v,j}$ are available and $F_{u,i}$ is the agreed fragment of $t_{u,i}$ **then**
    set the validity of $t_{u,i}$ to get_validity$(t_{u,i}, F_{u,j}, F_{v,j})$

We make two remarks. First, just like Π_t, we do not block any part of the protocol. Second, suppose some $F_{v,j}$ validates $t_{u,i}$, then that does not imply that $t_{u,i}$ only has one pair $t_{v,j}$. Our validity requirement only requires that there is only one $t_{v,j}$ in the correct consensus round. The counterparty may create any number of fake pairs in later consensus rounds. But these fake pairs only pollutes the chain of v and can never be validated.

IV. Design Variations and Tradeoffs

In this section, we explore a few design variations, some of them require a relaxed version of our original model. They enable better performance and allow us to apply our design in the fully permissionless setting.

A. Open membership using timing assumption

At the start of our consensus phase (Section III-B2), facilitators must wait for $N - f$ `cp_msg` messages. The use of N makes our system unsuitable for the open membership setting, where nodes may join and leave at will (churn). We over come this problem by introducing a timing assumption. Concretely, instead of waiting for $N - f$ messages, we wait for some time D, such that D is sufficiently long for honest nodes to send their CP blocks to the facilitators. Consequently, this removes the need for a PKI because the collected CP blocks may be from nodes that nobody has seen in the past.

The new protocol handles churn as follows. Suppose a new node wish to join the network and the facilitators are known (this can be done with a public registry). It simply sends its latest CP block to the facilitators. Then, in the next round, the node will have a chance to become a facilitator just like any existing node. To leave the network, nodes simply stop submitting CP blocks. There is a subtlety here which happens when the node is elected as a facilitator in the following round. In this case, the node must fulfil its obligation by completing the consensus protocol, but without proposing its own CP block, before leaving. Otherwise, the $n \geq 3t + 1$ condition may be violated.

B. Optimising Validation Protocol Using Cached Agreed Fragments

One more way to improve the efficiency of Π_v is to use a single agreed fragment to validate multiple transactions. Concretely, for node A, upon receiving an agreed fragment from node B, rather than validating a single transaction, A attempts to validate all transactions performed with B, which are in the unknown state but also in that fragment.

The benefit of this technique is maximised when a node only transacts with one other node. In this case, the communication of one fragment is sufficient to validate all transactions in that fragment. In the opposite extreme, if every transaction that the node makes is with another unique node, then the caching mechanism would have no effect.

C. Total Fork Detection

The validation algorithm guarantees that there are no forks within a single agreed fragment, which is sufficient for some applications such as proving the existence of some information. However, for applications such as cryptocurrency where every block depends on one or more previous blocks, our scheme is not suitable. For such applications, we need to guarantee that there are no forks from the genesis block leading up to the TX block of interest.

We offer two approaches to do total fork detection. First and the easiest solution is to ask for the complete hash chain

of the counterparty. The verifier can be sure that there are no forks if the following conditions hold.

1) The hash pointers are correct.
2) All the CP blocks are in consensus.
3) The TX of interest is in the chain.

We use this approach in our prior work on Implicit Consensus [21]. Nodes employ caching to minimise communication costs, and we call this effect spontaneous sharding.

The second approach is probabilistic but with only a constant communication overhead over our current design. For a node, observe that if all of its agreed fragments has a transaction with an honest node, then the complete chain is effectively validated in a distributed manner. The only way for an attacker to make a fork is to ensure that the agreed fragment containing the fork has no transactions with honest nodes. Such malicious behaviour is prevented probabilistically using a challenge-response protocol as follows. Suppose node A wish to make a transaction with node B. A first sends a challenge to B asking it to prove that it holds a valid agreed fragment between some consensus round specified by A. If B provides a correct and timely response, then they run the transaction protocol as usual. Otherwise, A would refuse to make the transaction.

D. Unbiased Facilitator Election

Our consensus protocol does not guarantee unbiased facilitator election when dedicated attackers are present. If a malicious facilitator is elected, it can delay, eavesdrop and collect all CP block messages before sending its own. Effectively, it can generate a CP block such that it has an unfair advantage of being elected as a facilitator in the next round.

To address the issue above, the facilitators run an extra protocol after the consensus protocol to produce some unbiased randomness. Concretely, they invoke RandHound [22] and then propagated the randomness and the signatures in the same way as the consensus result. Upon receiving the randomness, every node uses it in the hash function of Algorithm 2 (i.e. $\mathsf{H}(\text{randomness}||\mathcal{C}_r||pk \text{ of } x)$) to compute the new set of facilitators.

V. Correctness and Fault Tolerance Analysis

We evaluate our system analytically to ensure the desired properties (Definition 6 and Definition 7) hold. An informal argument is given in this section. We refer to [23, Chapter 4] for an in-depth analysis.

A. Correctness of the Consensus Protocol

Π_c correctly implements the CHECO consensus protocol (Definition 6) due to the following. The agreement, validity and termination properties hold because:

- The CP blocks sent to the facilitators are eventually delivered, and then ACS eventually starts.
- Agreement, validity and termination hold for ACS as they are the properties of ACS and are proven to hold in [12].
- The consensus result and signatures are eventually disseminated to all the nodes, so honest nodes must hold the same result as the honest facilitators.

B. Correctness of the Validation Protocol

Using the previous result, we show that Π_v implements the agreement and validity properties of a CHECO validation protocol (Definition 7).

The validity property holds because we use $\mathsf{get\_validity}(\cdot)$ in the validation protocol. The agreement property holds because we model $\mathsf{H}(\cdot)$ as a query to a random oracle. That is, suppose two honest nodes decided on two different states, *valid* and *invalid* for the same transaction. For that to happen, two agreed fragments must exist for the same transaction, but these fragments must also have the same agreed enclosure. Recall that blocks form a hash chain. So this is not possible unless the adversary can compute the inverse of $\mathsf{H}(\cdot)$ with high probability.

Liveness, unfortunately, does not hold in our model. A malicious node can act honestly when running the transaction protocol, but then never respond to any validation requests. Therefore some transactions can never be validated. Nevertheless, the malicious node will be at an economic loss if it is not responsive because honest nodes are less likely to make contact with nodes that do not respond to validation requests. If the probabilistic fork detection proposal (Section IV-C) is used, the uncooperative nodes will have more incentive to participate in the protocol.

VI. Implementation and Evaluation

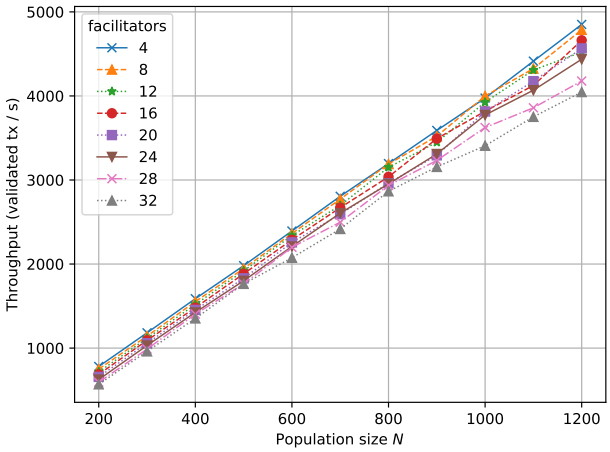
A free and open source implementation can be found on GitHub: https://github.com/kc1212/checo. It implements the three protocols and the Extended TrustChain. We also implement the caching optimisation discussed in Section IV-B. The cryptography primitives we use are SHA256 for hash functions and Ed25519 for digital signatures.

We run the experiment on the DAS-5[3] with up to 1200 nodes. Every node makes transactions at 2 per second. Since Bitcoin transactions are approximately 500 bytes [24], we use a uniformly random transaction size sampled between 400 and 600 bytes.

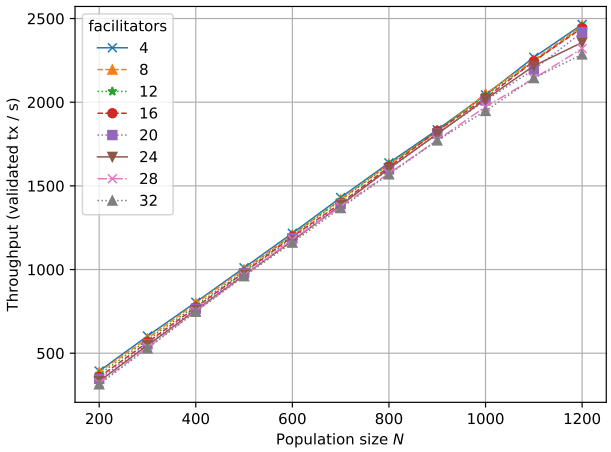
The global throughput results are shown in Figure 2. We consider Figure 2a as the ideal case, where nodes only make transactions with a fixed node. Figure 2b is the worst case, where nodes make transactions with random nodes and the caching mechanism is unlikely to be used. Observe that the transaction rate is much lower in Figure 2b, which is because the communication of an agreed fragment is necessary to verify every transaction (no caching), putting a strain on our network infrastructure.

For Figure 2a, the magnitude of our throughput may not be self-evident at first glance. Recall that we fixed the transaction rate to 2 TPS, but how is it possible to have around 4800 transactions per second for 1200 nodes (which is 4 TPS)? It is due to the way validated transactions are calculated. Transactions are between two parties, hence if every node makes two transactions per second, every node also expects to receive two transactions per second. Hence, for every node, the

[3]https://www.cs.vu.nl/das5/



(a) Every node make transactions with a fixed node.



(b) Every node make transactions with a random node.

Fig. 2: Global throughput increases as the population increases when every node transact at the same rate. Making transactions with fixed nodes results in a higher throughput because of the caching mechanism.

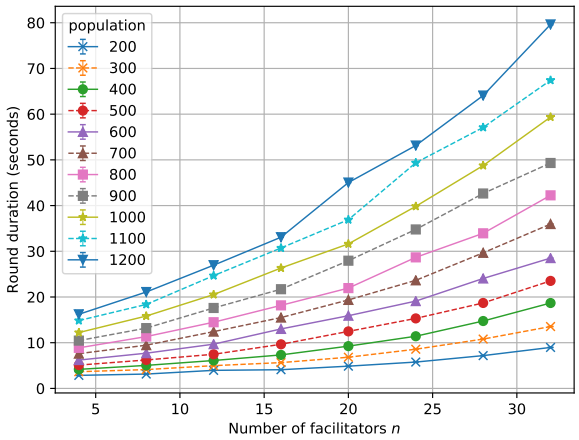


Fig. 3: The consensus duration increases polynomially with respect to the number of facilitators.

TX blocks are created at 4 per second. Validation requests are sent at the same rate, which explains the magnitude. Overall, the throughput has a linear relationship with the population size. This result is a strong indication of the horizontal scalability which we aimed to achieve.

The downside of our design is that the communication complexity of the consensus protocol grows polynomially with respect to the number of facilitators. Hence, the consensus protocol will take longer to complete, and larger fragments must be sent for transaction verification. On the other hand, it does not significantly impact the throughput; only the transaction verification delay is affected. The experimental results in Figure 3 demonstrates this issue, it uses the same experimental-setup as before. We refer the reader to [23, Chapter 5] for additional analysis of the effect of the number of facilitators as well as other experimental results.

VII. Conclusion

In this work, we described CHECO, an application neutral blockchain system with horizontal scalability. Our novel data structure allows nodes to efficiently store transactions and record state using CP blocks. The round based consensus protocol uses ACS as a building block to reach consensus on CP blocks. The consensus result lets nodes elect new facilitators and create new checkpoint blocks. To make transactions, nodes use the simple and non-blocking transaction protocol. Finally, we introduce a validation protocol which ensures that if an agreed fragment for some transaction exists, then nodes reach agreement on the validity of that transaction. The novelty of CHECO is that it decouples consensus and transaction validation, which enables the desirable horizontal scalability property, without employing sharding.

We achieve the properties described in Section II. Namely, our protocol achieves agreement on transactions as we argued in Section V-B. Validity is achieved because honest nodes run the get_validity$(\cdot)$ function, which, in fact, is the validity definition. Further, the horizontal scalability is demonstrated in Section VI, in the ideal case as well as the worst case.

In the future, we hope to apply our system to a concrete application and evaluate its performance. Furthermore, we plan to explore a few useful design alternatives such as open membership, total fork detection and fair facilitator election.

References

[1] CoinMarketCap. (Jun. 2017). Cryptocurrency market capitalizations, [Online]. Available: https : / / coinmarketcap . com / currencies / bitcoin/ (visited on 08/05/2017).

[2] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 17–30.

[3] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, USENIX Association, 2016, pp. 279–296.

[4] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger.," *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.

[5] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*, Springer, 2015, pp. 112–125.

[6] E. Lombrozo, J. Lau, and P. Wuille. (Dec. 2015). Segregated witness (consensus layer), [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (visited on 06/25/2017).

[7] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*, Springer, 2016, pp. 106–125.

[8] J. Poon and T. Dryja, "The bitcoin lightning network," Jan. 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf.

[9] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*, Springer, 2015, pp. 3–18.

[10] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[11] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[12] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 31–42.

[13] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus," *arXiv preprint arXiv:1612.02916*, 2016.

[14] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 251–260.

[15] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.

[16] Namecoin Developers. (). Namecoin, [Online]. Available: https://www.namecoin.org/ (visited on 06/25/2017).

[17] P. Otte, "Sybil-resistant trust mechanisms in distributed systems," Master's thesis, Delft University of Technology, Dec. 2016. [Online]. Available: http://resolver.tudelft.nl/uuid:17adc7bd-5c82-4ad5-b1c8-a8b85b23db1f.

[18] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips, "Tribler: A social-based peer-to-peer system," *Concurrency and computation: Practice and experience*, vol. 20, no. 2, pp. 127–138, 2008.

[19] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*, ACM, 1993, pp. 62–73.

[20] R. Wattenhofer, *Principles of distributed computing*, 2016. [Online]. Available: http://dcg.ethz.ch/lectures/podc_allstars/lecture/podc.pdf.

[21] Z. Ren, K. Cong, J. Pouwelse, and Z. Erkin, *Implicit consensus: Blockchain with unbounded throughput*, 2017. eprint: arXiv:1705.11046.

[22] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *Security and Privacy (SP), 2017 IEEE Symposium on*, IEEE, 2017, pp. 444–460.

[23] K. Cong, "A blockchain consensus protocol with horizontal scalability," Master's thesis, Delft University of Technology, Aug. 2017. [Online]. Available: http://resolver.tudelft.nl/uuid:86b2d4d8-642e-4d0f-8fc7-d7a2e331e0e9.

[24] TradeBlock. (Oct. 2015). Analysis of bitcoin transaction size trends, [Online]. Available: https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends (visited on 07/14/2017).

On the Delay Performance of Browser-based Interactive TCP Free-viewpoint Streaming

Tilak Varisetty, Markus Fidler
Institute of Communications Technology
Leibniz Universität Hannover
Email: firstname.lastname@ikt.uni-hannover.de

Matthias Ueberheide, Marcus Magnor
Computer Graphics Lab
Technische Universität Braunschweig
Email: lastname@cg.cs.tu-bs.de

***Abstract*—In free-viewpoint video arbitrary views of a scene or an object are rendered from a 3-dimensional scene representation that is obtained using multiple cameras or generated by computer graphics. The interactivity that is due to the viewpoint selection is particularly challenging in case of networked applications, where a server renders the scene from a viewpoint that is chosen by a remote client. Relying on widely-used standard browser-based video streaming technology, data transport is performed by the Transmission Control Protocol (TCP), implying an anticipated risk of potentially large delays. The magnitude, frequency, and origin of such delays are the focus of this work. To investigate the tail distribution of the delays, we use a controlled testbed environment and instrument the entire video streaming chain from the server-side renderer to the display at the client using various measurement points. We identify three major sources of delays: the video coders, the protocol stack, and the network. We investigate the causes of these delays and show a strong impact of network parameters, such as round-trip time and packet loss probability, on protocol stack delays. While stack delays can significantly exceed network delays, we find that stack delays can be reduced effectively by adapting the parameters of the video encoder.**

I. Introduction

Multiview video arises in many situations where a scene is captured simultaneously by multiple cameras from different viewpoints. The cameras can be configured as specific camera arrays, or they can be naturally located at various positions, e.g., cameras in a sports stadium or the cameras of mobile phones that may be used to record a public event. Applications of multiview video include immersive telepresence systems, 3-dimensional stereoscopic films, or free-viewpoint television, where the viewer can freely navigate the viewpoint [1]. The selected viewpoint may either coincide with a given camera position or otherwise it may be rendered using the views of nearby cameras. The rendering of a scene or an object from a given viewpoint is also performed in many other application areas of computer graphics such as gaming or Computer Aided Design (CAD).

In a video streaming application a server encodes the video and transmits it to one or more clients for simultaneous reproduction. Due to varying network latencies referred to as delay jitter, the client first stores the received data in a de-jitter buffer, and then displays it from the buffer after a defined playout delay that considers, e.g., a certain quantile of the network latencies.

This work was supported in part by the European Research Council (ERC) under StG 306644.

ISBN 978-3-903176-08-9 © 2018 IFIP

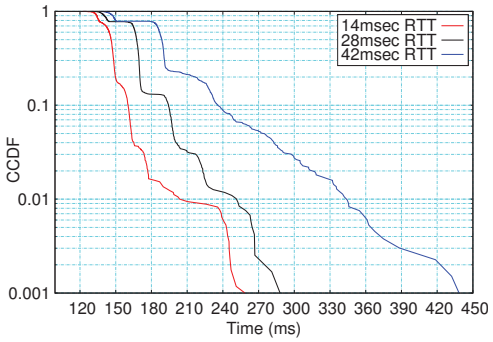


Fig. 1. End-to-end delay of interactive TCP streaming of a free-viewpoint application with server-side rendering. Different network RTTs of 14, 28, and 42 ms are evaluated. The packet loss rate is 1%. In the distribution tail, delays exceed the RTT by an order of magnitude.

Streaming of multiview video is particularly challenging due to the high bandwidth requirements when transmitting multiple video streams [2]. Compression techniques such as Multiview Video Coding (MVC) have been developed that use prediction to take advantage of the temporal correlation within the streams of the individual views (intra-view prediction) as well as the spatial correlation between different views (inter-view prediction).

Providing the entire set of all views to a client enables non-interactive multiview video streaming, where viewpoint selection or rendering of the viewpoint can be performed locally, i.e., by the client. A significant reduction of the data rate can be achieved, however, if only a selected subset of the available views needs to be transmitted. This is done in interactive streaming that uses a control channel from the client to the server to notify the server about the client's viewpoint selection [2]. Using this information, the server may either transmit a range of (potentially) relevant views, so that selection or rendering of the viewpoint remains with the client, or alternatively the server may perform the rendering itself and transmit only the single selected view [3]. Server-side rendering has the additional benefit that only a view but not the source data, for example a CAD model, need to be disclosed.

The reduction in bandwidth that is achieved by interactive streaming entails, however, more stringent latency requirements. Given a subset of the views, the client's viewpoint navigation is constrained by the boundaries of this subset [4], [5]. To prevent the client from selecting a viewpoint outside of this subset, the server has to continuously keep track of the viewpoint or even make a prediction of future viewpoints to adapt the range of views that it provides accordingly [5]–[7]. In case of server-side rendering, client and server have to interact in real-time, as the client's selection of the viewpoint has to be considered immediately by the server.

To explore the feasibility of interactive free-viewpoint streaming we investigate the delay performance of a reference implementation [8] in a controlled network testbed. We implement a server-side application that takes a 3-dimensional model to render selected viewpoints without disclosing the source data. We employ open source video coders and streaming servers and consider a browser-based implementation of the client that uses only standard HTML5 streaming without requiring any additional plugins. As HTML5 streaming is an emerging technology, the availability is still limited by constraints such as codec, browser, operating system, and type of video streaming [9]. The choice of HTML5 streaming also implies the use of the Transmission Control Protocol (TCP).

Fig. 1 shows an example of the Complementary Cumulative Distribution Function (CCDF) of the end-to-end delay of video frames from the server-side renderer to the display at the client. The end-to-end delays comprise the time needed for encoding and decoding of the video as well as the transmission of the video frames over the network. The measurements are performed using different network configurations with Round Trip Times (RTTs) of 14, 28, and 42 ms. Other parameters have not been changed and the packet loss rate is 1% in all experiments. The effect of the RTT is clearly visible in the upper part of Fig. 1, where the curves are spaced 14 ms apart from each other, as expected. A much stronger impact, that exceeds the RTT by an order of magnitude, is visible in the tail distribution of the end-to-end delays.

This paper reports the results of an extensive measurement study. We investigate the occurrence of large tail delays, show their causes and how they may be circumvented. To identify where delays occur, we instrument the video encoding and transmission chain and perform logging at various measurement points. Given that video frames are segmented into several packets, i.e., TCP segments, we implement a frame logging mechanism that identifies video frame boundaries in a packet stream with packet loss and retransmissions to be able to measure the delivery of entire video frames.

The remainder of the paper is structured as follows. In Sec. II we discuss related works on multiview and free-viewpoint streaming and the performance of TCP streaming. We show our experimental setup in Sec. III. Our measurement results obtained from the network experiments are presented in Sec. IV and Sec. V, where we investigate the impact of network and video parameters, respectively. Brief conclusions are provided in Sec. VI.

II. Related Work

We first discuss related works in multiview and free-viewpoint streaming and focus on the streaming performance of TCP afterwards.

A. Multiview and Free-viewpoint Video Streaming

Specific to multiview video streaming systems are multiple cameras, e.g., a camera array, that capture a scene from different viewpoints. The content that is generated by the cameras is encoded and streamed by a server over a network to one or more clients that can choose the viewpoint individually from the set of views that are available. Using techniques from computer graphics, the client may also render arbitrary new views, thus enabling a free-viewpoint navigation. Significant work has been dedicated to the quality and the complexity of the view synthesis. For an introduction to multiview video streaming see [2] and to free-viewpoint video [1].

One of the main challenges in multiview video streaming is the bandwidth that is required to transmit a potentially large set of views to the client. Various techniques have been developed to reduce the amount of data that is needed. In source coding, considerable works have taken advantage of the spatial correlation of the individual views to achieve a higher compression rate. A prominent example is the H.264/MPEG-4 extension MVC [10] that makes efficient use of temporal as well as spatial prediction. Using MVC, all views can be jointly encoded and transmitted, so that the client can choose the viewpoint without further interaction with the server.

Interactive multiview streaming, on the other hand, uses a control channel from the client to the server to report the viewpoint that is chosen by the client [2]. Using this information, the server can transmit only the selected view to the client to save bandwidth. Switching to a different view implies a random access that can be supported by insertion of intracoded frames. Since intracoded frames achieve less compression gain, more efficient frame structures, so-called merge frames, that enable view switching at defined intervals T, are presented in [11].

A concern with interactive multiview streaming is the view switching delay, that occurs if the user switches to a new view that is not streamed currently. In addition to the view switching interval T that can be small, i.e., in the order of a few frames [5], the network RTT of up to hundreds of milliseconds may have a considerable impact on the view switching delay [5], [12]. An approach to avoiding view switching delays is transmitting additional views, possibly with a higher compression [7], that are likely to be requested by the client within one RTT [4], [5], [7]. This creates a general tradeoff between bandwidth and latency [12] and requires a good anticipation of the viewpoint navigation of the user [5], [6]. Similar aspects concern free-viewpoint video streaming, where rendering can be performed at the client or at the server, requiring either sufficient bandwidth to transmit multiple views or small RTTs, respectively [3].

The importance of the RTT for view switching in interactive video streaming is emphasized in [2]–[5], [12]–[14]. While

a number of works use an assumption of a deterministic RTT [4], [5], [7], mostly as a constant to determine a range of viewpoints that need to be prefetched, real networks exhibit large delay variations, as observed, e.g., in experiments in [12]. Further, recent works employ Dynamic Adaptive Streaming over HTTP (DASH) and TCP for free-viewpoint streaming with client-side rendering [14] and for multiview streaming [13], where delays are observed that exceed the RTT by orders of magnitude. For an example, [13] implements a number of techniques like buffer control, server push schemes, and parallel streaming to reduce the average view switching delay from 3.2 s to 380 ms given a network RTT of 4 ms.

B. TCP Streaming Performance

Despite the fact that TCP may cause large delays, it has become popular for streaming for other reasons, e.g., to circumvent firewalls. Extensive literature is available on the performance of TCP in multimedia streaming and specifically DASH. Relevant performance measures include the throughput and different types of delays that impact the users' Quality of Experience [15].

A major source of difficulty in TCP streaming is the variability of TCP's throughput. A common approach is to adapt the data rate of the encoded video to react to fluctuations of the available network bandwidth [16]. To reduce delays, [17] improves the adaptation based on TCP throughput predictions. To what extent the achievable TCP throughput can be utilized at all is investigated using an analytical model in [18]. An important conclusion is that good streaming performance is attained when the achievable TCP throughput is at least twice the video data rate.

Regarding TCP delays, different types of delays have to be distinguished. While there exist numerous studies that investigate network delays of TCP packets, fewer works have focused on TCP protocol delays [19]–[25]. TCP protocol delays are defined as the time difference from a write on the sender side socket to the corresponding read on the receiver side socket [19]. They comprise network delays plus potential transport layer queueing delays at the sender and at the receiver [23].

The works [19]–[22] present Cumulative Distribution Functions (CDFs) of protocol delays that for certain relevant network parameters show a long distribution tail due to heavy sender-side buffering. The authors of [19], [20] conclude that large tail delays may be avoided by reducing the sender's socket buffer size. Based on a testbed measurement study, [21] develops a parametric model of the CCDF of TCP protocol delays. The CCDFs show a characteristic exponential tail decay that depends on the relation of the average TCP throughput and the source data rate. The importance of this relation was already discussed above as it is also observed in [18].

An analytical model of TCP delays is derived for Constant Bit Rate (CBR) sources and the TCP version NewReno in [23]. The model gives working regions, i.e., RTTs and packet loss rates, that provide acceptable delay performance for streaming applications. The work [22] estimates service curves of different TCP versions and shows how the different algorithms for adaptation of the Congestion Window (CWND) affect the video streaming performance. Queueing models of TCP's finite state machine are used to analyze its performance in [26], [27].

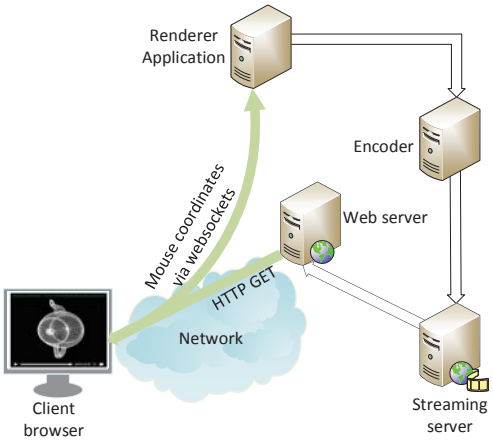


Fig. 2. System overview. The renderer application generates images at a fixed frame rate using the viewpoint that is selected by the mouse coordinates of the client. The images are encoded and streamed to the client browser by a webserver using HTTP and TCP.

III. Experimental Setup

In today's Internet, TCP became a de facto standard for video streaming that has recently been adopted for more demanding applications such as multi-view [13] and free-viewpoint video streaming [14]. While [13], [14] use DASH with view switching delays of several hundreds of milliseconds up to seconds and client-side free-viewpoint rendering [14], we consider an interactive free-viewpoint video streaming system with server-side rendering that has significantly more demanding delay requirements. Like [13], [14] we implement a browser-based solution that also uses TCP but not DASH for streaming. In this section, we first give a brief overview of our implementation of free-viewpoint video streaming [8]. We then introduce our Emulab testbed configuration that we use for experimentation, and give details on our instrumentation of the application and network that enables us to distinguish different causes of delays.

A. Free-viewpoint Video Application

An overview of the system that we implemented is given in Fig. 2. The setup uses a client-server model where the server executes the necessary software for rendering, encoding, streaming, and hosting of the video. The view that is generated by the server-side renderer application is selected by the client.

The renderer application is programmed using OpenGL libraries. It uses libwebsockets on a specified TCP port to receive the coordinates of the viewpoint, i.e., the mouse coordinates, from the client. The renderer application periodically creates new images based on the selected viewpoint. Rendering is performed at a configurable rate of up to 50 frames per second (fps).

TABLE I
SOFTWARE VERSIONS.

Software	Version No.
Ubuntu	12.04.5 LTS
ffmpeg, ffserver	2.1.git
lighttpd	1.4.28
OpenGL	2.1 Mesa 8.0.4
libwebsocket	1.0.8
libvpx	1-3.0

Whenever the renderer creates a new image, it is encoded by the open source software ffmpeg using the libvpx video codec. The encoded image is streamed by ffserver and made available to the client as an HTML5 file by a lighttpd webserver. The client sends an HTTP GET request to receive the HTML5 stream from the webserver. Streaming is performed via TCP, specifically the TCP version Cubic with selective acknowledgements (SACK) option. The client runs Google's Chrome browser to play the video. As our setup only uses standard HTML5 streaming technology, the client's browser does not require any plugins.

For the purpose of the following measurements, we have configured the video encoder to produce constant-sized frames, specifically intracoded frames, to ease the interpretation of the results by avoiding delay variations that are due to the size of the frames. The size of video frames is approximately 12.6 kByte if not specified otherwise.

Tab. I summarizes the versions of the software used. Further details on our implementation can also be found in [8].

B. Network Testbed Configuration

We use the Emulab installation at our institute to evaluate the performance of our free-viewpoint video streaming application. Emulab[1] is a well-known framework for network emulation that can configure arbitrary network topologies consisting of nodes, i.e., hosts and routers, and links for controlled experimentation. Each node is put into effect by a physical machine that is booted with a defined operating system to act either as a router or as a host that runs the intended network application.

The machines have several network interfaces for experimentation, in our case a minimum of four 1 Gbps Ethernet interfaces, that are connected to a central switch. The switch creates Virtual LANs (VLANs) between the nodes to form the desired topology. Link parameters such as capacity, delay, and packet loss are emulated by the system using additional nodes, e.g., a link with a fixed delay comprises two VLAN links connected to a delay node that forwards packets only after the defined amount of time. The ipfw utility on freebsd is used for this purpose[2]. Capacity limits and packet loss are emulated in the same way. The machines have additional interfaces that are connected to a separate control network that is used to execute and monitor the experiments and to synchronize the clocks of the different machines using the Network Time Protocol (NTP) and the institute's NTP server.

[1]https://www.emulab.net/

[2]https://www.freebsd.org/cgi/man.cgi?ipfw(8)

TABLE II
UTILIZATION WITH RESPECT TO THE GREEDY TCP THROUGHPUT FOR A VIDEO FRAME RATE OF 10 FPS, DIFFERENT RTTS, AND LOSS RATES.

Loss rate (%)	RTT (ms)	Throughput (Mbps)	Utilization (%)
0.5	28	5.03	20.0
1	14	7.29	13.8
1	28	3.57	28.2
1	42	2.58	39.0
2	28	2.57	39.1

In our experiments, we use 1 Gbps Ethernet links to connect the video client and server. The Maximum Transmission Unit (MTU) of the Ethernet is 1500 Byte such that the TCP Maximum Segment Size (MSS) is 1460 Byte and video frames of 12.6 kByte size result in 9 TCP segments. The central link is configured to emulate a wide area network with RTTs of 14, 28, and 42 ms, respectively. In addition, the link has independent Bernoulli random packet losses of 0.5, 1, and 2%, respectively. Given the link parameters, the throughput of a TCP connection is limited by TCP's congestion control algorithm. We measured the throughput that is achieved by a greedy TCP Cubic source for these network parameters using iperf[3]. The results are detailed in Tab. II. We define the utilization of the TCP connection by the video source as the quotient of the video data rate and the greedy TCP throughput. The greedy TCP throughput is the maximal throughput of the TCP connection, that is achieved in case of a greedy data source. Tab. II gives the utilizations for a frame rate of 10 fps. The utilization for other rates of x fps follows by multiplication with $x/10$.

C. Measurement Points and Delays

To identify and evaluate the causes of large end-to-end delays, as observed in Fig. 1, we instrument the video application and the network using a number of Measurement Points (MP) as shown in Fig. 3. The trace files that are recorded contain tuples of timestamp and value, where the value is either a unique video frame identifier or an entire TCP segment, depending on whether the MP is in the application or in the network.

In detail, MP-A denotes the timestamp after an image has been created by the renderer application glrender. The image is written to a pipe file and MP-B denotes the time when ffmpeg fetches the image from the pipe. After encoding, ffmpeg writes the corresponding video frame to a localhost socket, that is MP-C. The frame is picked up from the localhost by ffserver at MP-D and written to the TCP socket where MP-E is the timestamp before the TCP send call. MP-F denotes the time when the first TCP segment of the video frame is transmitted on the network and MP-G is the time when the first TCP segment of the video frame arrived at the client. Once all TCP segments of the frame are received, the frame is delivered to the browser for reproduction at MP-H. Since TCP delivers data in order, the delivery of a complete frame is delayed if one of the preceding frames is not yet complete.

[3]http://software.es.net/iperf/

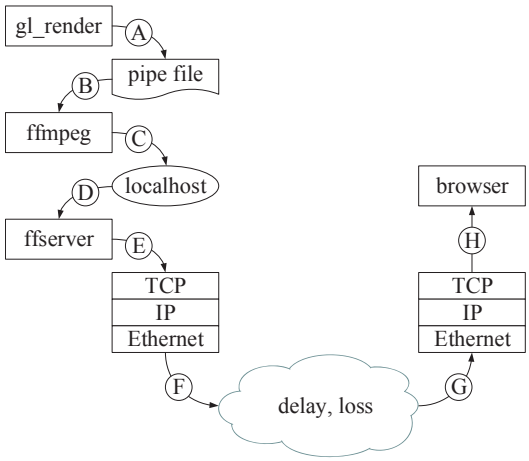


Fig. 3. The streaming pipeline is instrumented using measurement points A to H. We distinguish coding delays (A→C and D→E), stack delays (C→D and E→F), and transfer delays (F→H).

The logging framework is implemented in the source code of the software of the glrender application, ffmpeg, and ffserver. The glrender application generates a video frame number which is unique for every frame. It is mapped to the ffmpeg input and used for logging from MP-A up to MP-E. After MP-D, the ffserver assigns a unique Presentation Time Stamp (PTS) to the header of each video frame that is transmitted to the client. We note the correspondence of frame number and PTS at MP-E and use the PTS for identification of frames up to MP-H. Since TCP divides the video frames into smaller segments, the TCP segments are captured at MP-F and MP-G in the network using libpcap, respectively, the wireshark software[4]. We postprocess the network trace files to identify the video frame boundaries using the TCP sequence number and the PTS from the recorded TCP segments.

The different types of delays that each video frame incurs in the streaming pipeline from the rendering application to the client browser can be classified as follows:

Coding delays: The coding delays comprise the loading time of the image from the pipe (A→B), FFM encoder delays (B→C), and delays due to packaging into the FFM container format (D→E). In our measurements, the average delay between A→B is 18 ms, and B→C is 14 ms, respectively. The delay between D→E is dependent on the frame rate as we observed that ffserver generally buffers one video frame to send the current frame on the TCP socket. Hence, the delay is 100, 50, and 33 ms for 10, 20, and 30 fps, respectively. The coding delays observed in the experiments are not dependent on the network conditions.

Stack delays: Delays in the sender's protocol stack occur whenever the transmission is throttled by TCP congestion control so that frames have to wait for transmission in the sender's TCP stack (E→F), specifically in the socket buffer. In our experiments, the socket buffer is configured to have a size of 3 MByte. While the authors of [19], [20] argued that delays in the sender's TCP stack can be avoided by reducing the socket buffer size, we note that this approach shifts the problem to the next higher entity that would have to adapt accordingly. Precisely, if the socket buffer is full, the write call to the TCP socket by ffserver (E) blocks so that ffserver cannot fetch further frames from the localhost interface causing additional delays there (C→D). In the following evaluation we will generally show the combined stack delays of each frame (C→D plus E→F).

Transfer delays: The transfer delay of a video frame comprises the time to transmit all packets of the frame via the network (F→G). In case of packet loss, retransmissions are required in addition that can cause additional waiting times at the receiver until all packets are received in sequence and the frame can be delivered to the browser (G→H). In the evaluation we will show the entire transfer delay until the last TCP segment of the current video frame is delivered. (F→H).

[4]https://www.wireshark.org/

IV. Impact of Network Delays and Loss

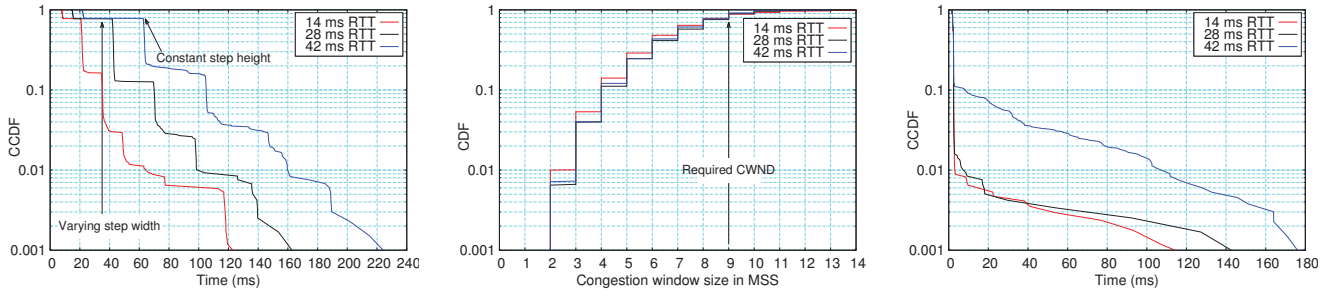
We conducted a large number of experiments with different network configurations and parameter sets. For clarity of exposition, we report experiments with selected parameter sets that enable us to separate and identify certain effects most clearly. In our evaluation, we first consider the impact of the network delay, specifically different RTTs of 14, 28, and 42 ms, on the transfer and stack delays experienced by the video frames. The network discards packets randomly to realize a loss rate of 1%. Different packet loss rates of 0.5% and 2% are considered afterwards. The frame rate is 10 fps and the frame size of 12.6 kByte corresponds to 9 TCP segments. The video bit rate at 10 fps is about 1 Mbps, i.e., compared to the greedy TCP throughput reported in Tab. II the utilization is moderate in all cases with a maximum of 39.1%.

A. RTT-induced Delays

The network RTT has an obvious impact on the transfer delay as it takes at least RTT/2 to deliver a video frame to the client. Frequently, the transfer delay is, however, larger as not all packets of a frame may be transmitted at once due to TCP congestion control. Further, packets may require retransmission in the case of packet loss. In addition, the RTT may result in stack delays and blocking of the sender's TCP socket.

1) Per-frame Transfer Delays: In Fig. 4(a), we show the dependence of the CCDF of the transfer delay (F→H) on the RTT. We notice that all curves show the same stepped trend, where the first step at 7, 14, and 21 ms, respectively, corresponds to RTT/2 and the following steps have a width of 14, 28, and 42 ms, respectively, corresponding to the RTT. The step height, on the other hand, shows little influence of the RTT and is almost identical for the first steps.

The behavior is explained by the CDF of the CWND that is shown in Fig. 4(b). First, we notice that the CDF of the CWND exhibits no significant influence of the RTT. As the CDF indicates the probability that the CWND does not exceed a given value, we see that a CWND of less than 9 MSS, that



(a) Transfer delay (b) CWND (c) Stack delay

Fig. 4. Impact of the RTT. The packet loss rate is 1%. The CCDF of the transfer delay exhibits first a minimum delay of RTT/2 followed by steps of a width of RTT. The CDF of the CWND shows only a marginal influence of the RTT. A CWND of at least 9 MSS permits sending a video frame at once. The probability that the CWND falls below 9 MSS corresponds to the probability to see transfer delays of more than RTT/2. Stack delays are observed when the transfer of a frame takes longer than 100 ms so that the next frame, at a frame rate of 10 fps, has to wait for transmission.

is the minimum CWND that is required to be able to transmit a video frame at once without waiting for acknowledgements, occurs with probability 0.77. In this case only a part of the video frame, i.e., CWND packets, can be transmitted before the transmission is paused. The transmission is resumed after one RTT when the first acknowledgements appear at the sender. We observe that the probability of 0.77 corresponds to the first step of a width of RTT in Fig. 4(a). The following steps can be explained in a similar way, e.g., in case of a CWND of less than 5 MSS it takes another round of one RTT. In addition, retransmissions that also take one RTT start to have an influence. In the given case of 9 TCP segments per frame and 1% packet loss, the transfer of a frame requires retransmission of one or more packets with a probability of almost 0.09.

We conclude that the RTT has only a minor effect on the CDF of the CWND in our experiments. Hence, the probabilities of observing transfer delays of several RTTs are similar irrespective of the RTT. In contrast, the magnitude of the transfer delays grows linearly with the RTT.

2) Stack Delays and Blocking: If the transfer of one or more video frames is not completed when the next video frame is ready for transmission, additional buffering applies at the sender resulting in stack delays and possibly blocking of the sender's TCP socket. Given the frame rate of 10 fps, this applies if video frames do not complete transfer within 100 ms. From Fig. 4(a), we find that transfer delays exceed 100 ms with a probability of slightly less or more than 0.01 in case of an RTT of 14 and 28 ms, and about 0.1 in case of an RTT of 42 ms. The CCDFs of the stack delays (C→D plus E→F) presented in Fig. 4(c) confirm these probabilities. Again, the effect is due to the probability that the CWND falls below a critical value. This critical value depends, however, on the RTT, so that the probability of incurring stack delays is also RTT dependent. For example, assume that the CWND is small, e.g., less than 5 MSS, so that it takes 3 rounds to deliver a video frame. In case of an RTT of 42 ms but not in case of 28 or 14 ms the transfer delay exceeds 100 ms and the next frame has to wait in the protocol stack.

The stack delays may also be interpreted as queueing delays at a system with random service [22] and the magnitude of the delays can be related to the utilization. Since the greedy TCP throughput depends on the RTT, see Tab. II, we have different utilizations of 13.8%, 28.2%, and 39.0% in case of an RTT of 14, 28, and 42 ms, respectively. We note that the stack delays are significant already at a moderate utilization of 39.0%.

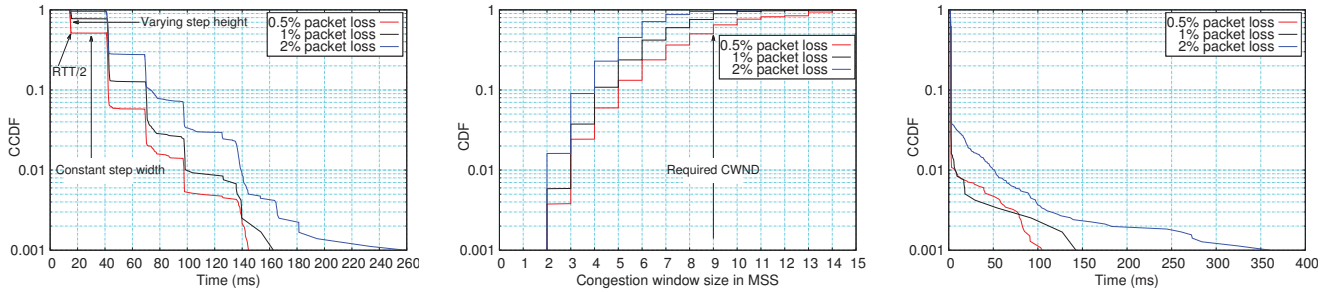
B. Loss-dependent Delay Probabilities

Next, we evaluate the impact of the packet loss probability. For the experiments, we set the loss rate to 0.5, 1, and 2%, respectively. The RTT is fixed to 28 ms.

1) Role of the CWND Distribution: Fig. 5(a) displays the CCDFs of the transfer delays. All curves show the same characteristic stepped shape as in Fig. 4(a) with a step width that is determined by the RTT of 28 ms. The curves differ, however, with respect to their step height that decreases in case of a larger packet loss rate, i.e., transfer delays that exceed the minimal transfer delay of RTT/2 by one or more RTTs become more frequent if the packet loss rate is larger.

The effect is caused by the CWND that is stochastically decreasing in the packet loss rate. The CWND falls with a higher probability below certain critical values if the packet loss rate is increased. The CDFs of the CWND are presented in Fig. 5(b). As before, a CWND of at least 9 MSS is required to be able to transmit an entire video frame at once, whereas it takes at least one additional round of one RTT if the CWND falls below 9 MSS. This happens with probability 0.5, 0.77, and 0.95 in case of a packet loss rate of 0.5, 1, and 2%, respectively. The probabilities correspond to the step heights of the first step of the transfer delay CCDFs in Fig. 5(a). The following steps are explained similarly.

2) Effect on Stack Delays: The CWND distribution also affects the probability and the magnitude of stack delays. With increasing packet loss rate, the CWND falls more frequently below the critical value that is required to deliver a frame within the frame generation interval of 100 ms. Likewise, the utilization increases with the loss rate, i.e., in the experiments the utilization is 20.0%, 28.2%, and 39.1% given the packet



(a) Transfer delay (b) CWND (c) Stack delay

Fig. 5. Impact of the packet loss rate. The RTT is 28 ms. The packet loss rate determines the height of the steps of the transfer delay CCDF. The effect is via the CWND, that is stochastically decreasing with the packet loss rate. For increasing packet loss rates, large tail delays are observed in the stack.

loss rates of 0.5%, 1%, and 2%, respectively, causing larger queueing delays in the stack.

V. Adaptation of Video Parameters

The RTT and the packet loss rate determine the TCP throughput as well as transfer and stack delays, as observed for a defined video traffic profile in Sec. IV. In order to mitigate delays, the video traffic can be adapted using a variety of encoding parameters that determine the compression gain and hence the data rate. Common options are adaptation of the temporal resolution, the spatial resolution, or the quantizer, i.e., the quality. While the first option determines the frame rate, the other two options influence the frame size. Generally, the goal of adapting the video data rate is to control the utilization to avoid load-dependent delays. In our evaluation, we discover that the interaction with the TCP protocol stack causes a number of relevant other effects.

A. Frame Rate

To investigate the impact of the frame rate, we adapt the renderer application to generate 10, 20, and 30 fps, respectively. First, we consider the effect of the utilization on delays before we turn to an artifact that is caused by TCP's fast retransmit algorithm.

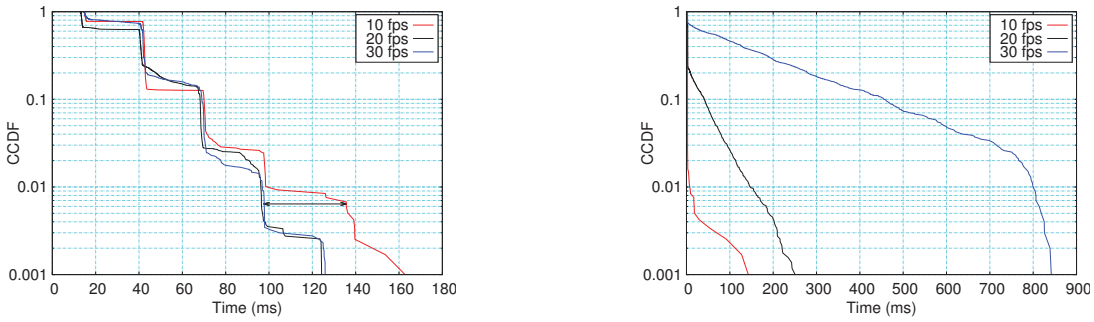
1) Effect of the Utilization: In Figs. 6(a) and 6(b) we show the impact of the frame rate on the transfer and the stack delay, respectively. The RTT is 28 ms and the loss rate 1%. In relation to the greedy TCP throughput the utilization is 28.2%, 56.4%, and 84.6% for 10, 20, and 30 fps, respectively. While we notice little change in the transfer delay in Fig. 6(a), a significant impact of the frame rate on the stack delay is observed in Fig. 6(b). As before, the stepped transfer delay curves in Fig. 6(a) are caused by the size of the CWND, e.g., if the CWND falls below the frame size of 9 MSS it takes one or more additional RTT-sized rounds to transfer a frame, see Fig. 4(b) and the explanation in Sec. IV-A1. This effect is independent of the frame rate.

If the transfer of a frame exceeds the time until the next frame is ready for transmission, that is the reciprocal of the frame rate, stack delays occur. Fig. 6(b) confirms that the probability of non-zero stack delays corresponds to the probability to see transfer delays of more than 100, 50, and 33 ms for 10, 20, and 30 fps, respectively. The magnitude of the stack delays in Fig. 6(b) demonstrates the effectiveness of rate adaptation and supports earlier observations [18], [22] that good delay performance is achieved only if the video data rate is smaller than the greedy TCP throughput by a factor of about two or more.

2) Issue of Last Segment Lost: In Fig. 6(a), the tail delay at a probability of 0.01 shows an opposing effect: the transfer delay is larger by almost 50 ms in case of a smaller frame rate of 10 fps. The effect persists if the RTT is reduced from 28 ms in Fig. 6(a) to 14 ms in Fig. 7, where additionally a difference of about 17 ms is noticed between the curves obtained for frame rates of 30 and 20 fps.

The reason for this is packet loss, specifically loss of the last TCP segment of a video frame. In general, TCP's fast retransmit algorithm, that is triggered by three duplicate acknowledgements, as well as selective repeat of missing segments indicated by SACKs deal effectively with loss. An exception is the loss of the last TCP segment of a video frame after which the transmission pauses. It resumes when the next video frame is available and eventually duplicate acknowledgements or a SACK indicate the missing TCP segment and trigger the retransmission. In the experiments, the loss rate is 1% and the frame generation period is 100, 50, and 33 ms in case of a frame rate of 10, 20, and 30 fps, respectively. Consequently, if the frame rate is smaller, it takes longer until the next frame resumes transmission and the retransmission is triggered. The time differences $100-50=50$ and $50-33=17$ ms are observed in Fig. 7 roughly at the loss probability of 0.01.

In more detail, if the last TCP segment of a frame is lost, it takes up to one frame generation period plus at least one RTT until duplicate acknowledgements or a SACK appear at the sender. Once there are three duplicate acknowledgements or a SACK, the retransmission is triggered. It arrives earliest after another RTT/2 at the receiver. In case of an RTT of 14 ms the numbers add up to 54, 71, and 121 ms for a frame rate of 30, 20, and 10 fps, respectively. The arrows in Fig. 7 mark these numbers approximately. For Fig. 6(a), where the RTT is 28 ms, the numbers are 75, 92, and 142 ms. The reason why



(a) Transfer delay

(b) Stack delay

Fig. 6. Impact of the frame rate. The RTT is 28 ms and the packet loss rate 1%. The frame rate determines the utilization that is 28.2%, 56.4%, and 84.6% for 10, 20, and 30 fps, respectively. The CCDF of the transfer delay shows little influence of the frame rate with significant deviations only in the tail. In contrast, the frame rate has a major impact on the stack delays. The probability to see non-zero stack delays corresponds to the probability that the transfer delay exceeds the frame generation interval of 100, 50, and 33 ms, respectively.

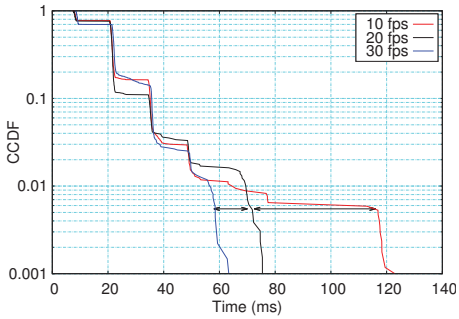


Fig. 7. Impact of the frame rate on the transfer delay. The RTT is 14 ms and the packet loss rate 1%. The tail transfer delays increase with decreasing frame rate. The effect is due to packet loss of the last TCP segment of a video frame. In this case duplicate acknowledgements or SACKs trigger TCP fast retransmit or selective repeat, respectively, only when the next frame is transmitted. The arrows indicate the differences between the frame generation intervals of 100, 50, and 33 ms.

the transfer delays in Fig. 6(a) show no difference between 30 and 20 fps is that due to the small CWND at probability 0.01, the tail delay to deliver a frame is 3.5 RTT, i.e., 98 ms, and hence larger than the estimated delays of 75 and 92 ms for retransmission of the last TCP segment.

B. Frame Size

Given that a reduction of the video frame rate can lead to larger transfer delays, we now investigate whether an adaptation of the frame size is more effective. We modify the quantizer to generate video streams with different frame sizes, where we specify the frame size relative to the one that we used in the previous experiments. We show results for frame sizes of 80%, 100%, and 120%. The frame rate is 10 fps and the network has an RTT of 28 ms and a loss rate of 1%.

1) Impact on the Transfer Delays: In Fig. 8(a), we consider the CCDF of the transfer delay. As before, we notice that the curves have distinct steps. The width of the steps is independent of the frame size and determined by the RTT. The height of the steps increases with decreasing frame size. This means that reducing the frame size effectively improves the transfer delay.

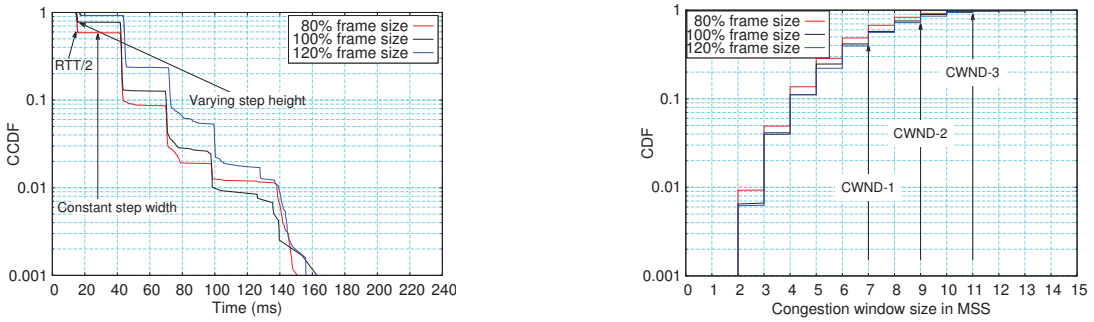
2) Relation with the CWND: The way in which the frame size helps reduce the transfer delay is via its relation to the CWND. First, we notice that the frame size has little effect on the CDF of the CWND, that is presented in Fig. 8(b), where larger frame sizes tend to result in slightly larger CWNDs. A possible reason for this is TCP's Congestion Window Validation algorithm [28], that freezes the CWND if it is not fully utilized. Hence, the CWND only rarely grows to large values if the frame size is small.

Given the similarity of the CWND CDFs in Fig. 8(b), the improvement of the transfer delay is due to the different CWND requirements given frames of different size. In the above case, the frame sizes correspond to 7, 9, and 11 TCP segments, respectively. If the CWND falls below any of these values, frames of the respective size cannot be transmitted in one round, resulting in one or more additional RTTs of transfer delay. The required CWNDs are marked in Fig. 8(b) and the probabilities that the CWND falls below any of these values are clearly visible as the height of the first step of the different transfer delay CCDFs in Fig. 8(a).

Regarding the stack delays, we observe relatively moderate values that do not show strong differences for the frame sizes above. The reason is the low utilization of 22.7%, 28.2%, and 33.8%, respectively. We omit showing the results and note, however, that the adaptation of the frame size can effectively reduce stack delays if the utilization is high.

VI. Conclusions

We investigated interactive TCP streaming of a free-viewpoint rendering application. We instrumented the entire streaming chain to identify delays and performed a comprehensive measurement study in a controlled network testbed to analyze the impact of relevant network and encoding parameters. A finding is that the utilization of the TCP connection has to be kept low as it has a major impact on delays in the sender's TCP socket buffer. Further, a pronounced influence of the distribution of the CWND on the transfer



(a) Transfer delay

(b) CWND

Fig. 8. Impact of the frame size. The RTT is 28 ms, the packet loss rate 1%, and the frame rate 10 fps. The frame size effectively controls the CWND requirements of the video stream, where a CWND of 7, 9, and 11 MSS, respectively, permits sending a video frame at once. The probabilities that the CWND falls below these critical values correspond to the probabilities to see transfer delays of more than RTT/2.

delay is noticed. The insights guide the design of adaptive applications and suggest to include feedback about the size of the CWND and the socket buffer filling to the application. Our measurements of the end-to-end delay showed a high variability of a few 100 ms. Hence, the receiver of a video streaming application has to provision an adequate de-jitter buffer to accomodate the observed tail delays. In case of our rendering application, frames are displayed immediately when they arrive at the receiver. For moderate RTT and packet loss, stalling is sporadic and the user perception of the time lag is mostly acceptable.

REFERENCES

[1] S. Aljoscha, "3d video and free viewpoint video – from capture to display," *Pattern Recognition*, vol. 44, no. 9, pp. 1958–1968, Sep. 2011.

[2] J. Chakareski, "Adaptive multiview video streaming: challenges and opportunities," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 94–100, May. 2013.

[3] A. Hamza and M. Hefeeda, "Adaptive streaming of interactive free viewpoint videos to heterogeneous clients," in *Procs. of the 7th International Conference on Multimedia Systems*, May. 2016, pp. 10–22.

[4] X. Xiu, G. Cheung, and J. Liang, "Frame structure optimization for interactive multiview video streaming with bounded network delay," in *Procs. of IEEE International Conference on Image Processing*, Sep. 2011, pp. 593–596.

[5] ——, "Delay-cognizant interactive streaming of multiview video with free viewpoint synthesis," *IEEE Trans. on Multimedia*, vol. 14, no. 4, pp. 1109–1126, Aug. 2012.

[6] T. Maugey and P. Frossard, "Interactive multiview video system with low complexity 2d look around at decoder," *IEEE Trans. on Multimedia*, vol. 15, no. 5, pp. 1070–1082, Aug. 2013.

[7] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, "Client-driven selective streaming of multiview video for interactive 3DTV," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1558–1565, Nov. 2007.

[8] M. Ueberheide, F. Klose, T. Varisetty, M. Fidler, and M. Magnor, "Web-based interactive free-viewpoint streaming: A framework for high quality interactive free viewpoint navigation," in *Procs. of the ACM International Conference on Multimedia*, Oct 2015, pp. 1031–1034, Short Paper.

[9] X. Yan, L. Yang, S. Lan, and X. Tong, "Application of HTML5 multimedia," in *Procs. of International Conference on Computer Science and Information Processing (CSIP)*, Aug. 2012, pp. 871–874.

[10] A. Vetro, T. Wiegand, and G. J. Sullivan, "Overview of the stereo and multiview video coding extensions of the H.264/MPEG-4 AVC standard," *Procs. of the IEEE*, no. 4, pp. 626–642, Apr. 2011.

[11] G. Cheung, A. Ortega, and N. M. Cheung, "Interactive streaming of stored multiview video using redundant frame structures," *IEEE Trans. on Image Processing*, vol. 20, no. 3, pp. 744–761, Mar. 2011.

[12] J.-G. Lou, H. Cai, and J. Li, "A real-time interactive multi-view video system," in *Procs. of the ACM International Conference on Multimedia*, Nov. 2005, pp. 161–170.

[13] D. Yun and K. Chung, "Dash-based multi-view video streaming system," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 99, no. 99, pp. 1–1, Apr. 2017.

[14] A. Hamza and M. Hefeeda, "A DASH-based free viewpoint video streaming system," in *Procs. of Network and Operating System Support on Digital Audio and Video Workshop*, Mar. 2014, pp. 55–60.

[15] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hofeld, and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 469–492, Mar. 2015.

[16] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Procs. of the ACM Conference on Multimedia Systems*, Feb. 2011, pp. 157–168.

[17] K. Miller, A.-K. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, pp. 41–44, Oct. 2016.

[18] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," in *Procs. of the ACM International Conference on Multimedia*, Oct. 2004, pp. 908–915.

[19] A. Goel, C. Krasic, K. Li, and J. Walpole, "Supporting low latency TCP-based media streams," in *IEEE International Workshop on Quality of Service*, Aug. 2002, pp. 193–203.

[20] A. Goel, C. Krasic, and J. Walpole, "Low-latency adaptive streaming over TCP," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 3, pp. 21–40, sep. 2008.

[21] R. Lübben and M. Fidler, "On characteristic features of the application level delay distribution of TCP congestion avoidance," in *Procs. of IEEE International Conference on Communications (ICC)*, May 2016.

[22] ——, "Service curve estimation-based characterization and evaluation of closed-loop flow control," *IEEE Trans. on Network and Service Management*, vol. 14, no. 1, pp. 161–175, Mar. 2017.

[23] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The delay-friendliness of TCP for real-time traffic," *IEEE/ACM Trans. on Networking*, vol. 18, no. 5, pp. 1478–1491, Oct. 2010.

[24] J. Wu, C. Yuen, and J. Chen, "Leveraging the delay-friendliness of TCP with FEC coding in real-time video communication," *IEEE Trans. on Communications*, vol. 63, no. 10, pp. 3584–3599, Oct. 2015.

[25] Y. Xiong, M. Wu, and W. Jia, "Rate adaptive real-time video transmission scheme over TCP using multi-buffer scheduling," in *Procs. of the International Conference for Young Computer Scientists*, Nov. 2008, pp. 354–361.

[26] R. L. Cigno and M. Gerla, "Modeling window based congestion control protocols with many flows," *Performance Evaluation*, vol. 36-37, pp. 289–306, Aug. 1999.

[27] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan, "Modeling short-lived TCP connections with open multiclass queuing networks," *Computer Networks*, vol. 44, no. 2, pp. 153–176, Feb. 2004.

[28] G. Fairhurst, A. Sathiaseelan, and R. Secchi, "Updating TCP to support rate-limited traffic," RFC 7661, Oct. 2015.

Alternative Handshake Mechanism for the Stream Control Transmission Protocol

Felix Weinrank, Irene Rüngeler, Michael Tüxen
Münster University of Applied Sciences
Department of Electrical Engineering and Computer Science
Stegerwaldstrasse 39
48565 Steinfurt, Germany
{weinrank,ruengeler,tuexen}@fh-muenster.de

Erwin P. Rathgeb
University of Duisburg-Essen
Faculty of Business Administration and Economics
Computer Networking Technology Group
Schützenbahn 70
45127 Essen, Germany
erwin.rathgeb@iem.uni-due.de

***Abstract*—The Stream Control Transmission Protocol (SCTP) is a message and connection oriented transport protocol using a cookie based four-way handshake for connection establishment. The intention of the four-way handshake is to provide a robust protection against flooding attacks, like TCP SYN-flooding. Although it protects the memory resources on the server side, current implementations require a quite large amount of CPU resources. For some of SCTP's use-cases, like the underlying transport protocol for Web Real-Time Communication (WebRTC) Data-Channels, SCTP's cookie protection is unnecessary and its four-way handshake delays the connection setup unnecessarily.**

We have developed an alternative handshake method which offers a more lightweight cookie exchange and a zero round-trip time (RTT) connection setup capability while still being fully backwards compatible with the existing handshake procedure. We describe the alternative handshake procedure and its advantages in common use cases like short living connections and WebRTC Data-Channels. In addition, we evaluate the alternative handshake's benefit to the regular handshake with measurements using our FreeBSD kernel implementation.

I. Motivation

Reducing the connection setup time has become a more and more important topic in the design of network protocols. With TCP Fast Open [1], TLS 1.3 [2] and QUIC [3], many popular protocols provide specific mechanisms to reduce the connection setup time. This development is driven by the perception that connection setup time has replaced insufficient bandwidth in the context of improving the user experience.

The Stream Control Transmission Protocol (SCTP) [4] is a reliable, connection oriented transport protocol using a cookie based four-way handshake to provide protection against INIT-flooding attacks, resulting in one round trip before the client can send the first data and two round-trips until the connection establishment is completed. Even though originally designed for the transmission of small signaling messages in the Signaling System No. 7 (SS7), SCTP's use cases have been extended over the last years, for example by Web Real-Time Communication (WebRTC) where SCTP is used as the underlying transport protocol for Data-Channels [5].

We have analyzed the regular SCTP handshake in multiple scenarios and by using the FreeBSD and Linux SCTP stacks. Previous work [6],[7],[8],[9] and our evaluation shows that the handshake procedure is a resource consuming operation and vulnerable to byte amplification attacks, which is covered by the measurement and evaluation section in detail.

ISBN 978-3-903176-08-9

In case of WebRTC Data-Channels, SCTP's four-way handshake is redundant and adds an unnecessary delay because the SCTP communication is encapsulated within the Datagram Transport Layer Security (DTLS) [10] protocol which already authenticates the peer during its handshake.

Therefore, we developed an alternative handshake procedure focused on a more lightweight and flexible method which addresses the requirements and approaches of today's transport protocols. Our main goal was to reduce the time to set up a transport connection, while still being fully backwards compatible with the regular handshake procedure defined in RFC4960 [4].

Before we introduce our approach of the alternative handshake in detail, we explain the regular handshake and show its drawbacks with respect to setup time and resource consumption. The measurement and evaluation section points out the benefits of the new extension by running tests and benchmarks on real hardware with our FreeBSD kernel implementation. In the next section we describe the interaction with existing SCTP extensions and how the new extensions have been implemented and integrated in the existing socket API. The last section summarizes this paper and gives an outlook on further research activities.

II. Regular Handshake

An SCTP association between a server and a client is established by a four-way handshake as shown in Figure 1. The client sends an SCTP message containing an INIT chunk to an SCTP server to initialize the association setup. The INIT chunk, as shown in Figure 2, consists of several parameters carrying a variety of information. Some of the parameters are mandatory in every INIT and INIT-ACK chunk, while other parameters are optional.

The mandatory parameters include the *Initiate Tag*, *Advertised Receiver Window*, number of incoming and outgoing streams and the *Initial Transmission Sequence Number* (TSN). The optional parameters provide a flexible way to store arbitrary information in those chunks, for example a list

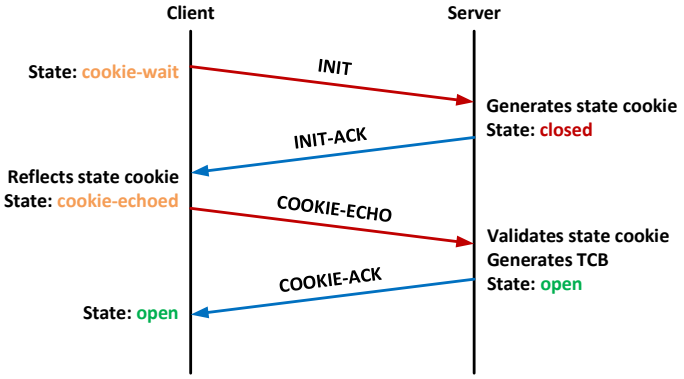


Fig. 1: Regular four-way handshake

of available IP address candidates and additional supported extensions.

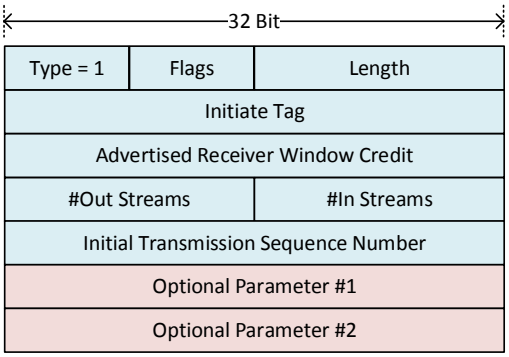


Fig. 2: INIT chunk structure

When receiving an SCTP message containing an INIT chunk, the server responds with an INIT-ACK chunk which consists of the same mandatory parameters as the INIT chunk and an additional *State Cookie* parameter. The state cookie contains all information the server needs to create the *Transmission Control Block* (TCB) and establish the association. It is digitally signed by the server to ensure its validity and neither the format nor the size of the cookie is specified by RFC4960, but it should be as small as possible. Since all necessary information to create the TCB is included in the cookie, the server does not need to allocate any client specific resources after the state cookie has been transmitted to the client. This mechanism is a key feature to prevent SCTP INIT-flooding attacks, similar to the TCP SYN-flooding, where an attacker can exhaust the server's resources. On the other hand, this can also be a drawback if the server stores a lot of information in the cookie, for example a long list of supported extensions or many address candidates. A large cookie size may be abused by an attacker for a byte amplification attack or to exhaust the server's uplink capacity. Byte amplification attacks cause the server to send a significantly larger INIT-ACK chunk as a response to an INIT chunk sent by an attacker with a spoofed source address of a victim. In addition to the state cookie, the server can also add any number of optional parameters to the INIT-ACK chunk.

Upon receiving the INIT-ACK chunk, the client returns the received state cookie to the server in a COOKIE-ECHO chunk to authenticate its ownership of the IP address. The server validates the reflected state cookie, creates the TCB, establishes the SCTP association and acknowledges the successful process with a COOKIE-ACK chunk. This COOKIE-ACK chunk opens the association on the client side, the four-way handshake has now successfully finished.

To reduce the timespan between connection initialization and transmitting the first data, the client and the server can bundle DATA chunks with the COOKIE-ECHO chunk and the COOKIE-ACK chunk, respectively, in a single SCTP message, resulting in a timespan of one round-trip for a client between initiating the connection and sending the first data.

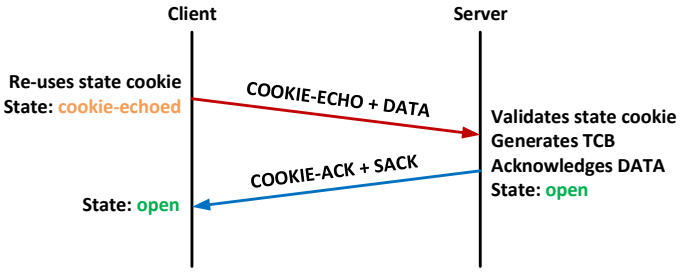


Fig. 3: Zero-RTT connection setup by re-using the state cookie from a previous association

In theory, the client can re-use the state cookie multiple times for future associations with the advantage of saving one round-trip for the connection setup, resulting in a zero-RTT connection setup, which is shown in Figure 3. A disadvantage of re-using the regular state cookie is that all connection specific parameters from the previous association, exchanged by the INIT and INIT-ACK chunk, have to stay the same which affects the *Initiate Tags* and the *Initial Transmission Sequence Numbers* in particular. Re-using the same *Initiate Tag* for more than one association is in conflict with its purpose: providing a key that allows a receiver to verify that the SCTP message belongs to the current association and is not an old or stale message from a previous association and should therefore be unique for every SCTP association. Additionally, the client cannot modify the supported extensions, number of streams or announced IP address candidates. Because of the drawbacks, this method is not implemented in the common network stacks.

III. Alternative Handshake

With our alternative handshake approach, we wanted to create a more lightweight cookie exchange mechanism which reduces the resource consumption on the server side and allows a faster connection setup in less round-trips compared to the regular handshake. Additionally, our alternative handshake mechanism should still be fully backwards compatible to systems without support for the new mechanism.

The alternative handshake, as shown in Figure 4, uses the same SCTP chunks as the regular handshake, only adding new optional parameters. A client using the alternative handshake method initiates an SCTP association by sending a regular INIT chunk to the server. The only difference to a regular handshake is the new ALT-COOKIE parameter which is included in the INIT chunk. By adding this parameter to the INIT chunk,

the client announces the support for the alternative handshake and its willingness to use it.

A server, also supporting the alternative handshake extension, who receives the alternative cookie parameter in the INIT chunk, generates a client specific cookie and includes it in an ERROR chunk. The ERROR chunk is sent as a response to the INIT chunk which is silently dropped by the server. This is the first major difference to the regular handshake: instead of generating an INIT-ACK chunk, containing all information the server needs to create the TCB, the server only generates a lightweight cookie to validate the IP address ownership of the client, which is the only purpose of the alternative cookie. The ERROR chunk has a specific error cause ("ALT-COOKIE required") and the client specific cookie in the *Cause-Specific Information* field.

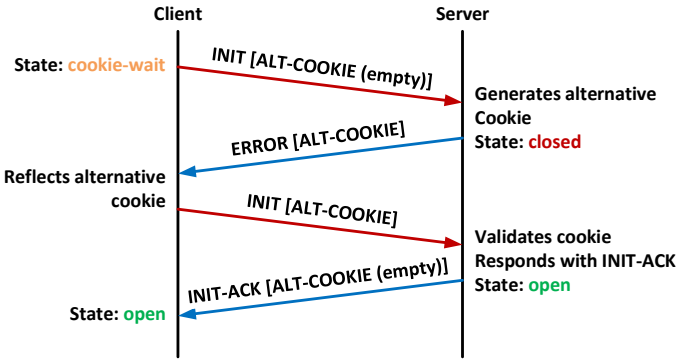


Fig. 4: Alternative four-way handshake

The client responds to the ERROR chunk by transmitting the INIT chunk and, in contrast to the first attempt, including the cookie from the server's ERROR chunk in the INIT chunk's ALT-COOKIE parameter payload field. Upon receiving the INIT chunk with a non-empty ALT-COOKIE parameter, the server validates the alternative cookie and opens the association if successful. In the next step, the server responds with an INIT-ACK chunk which carries an empty ALT-COOKIE parameter and no state cookie to acknowledge the successful usage of the alternative handshake to open the association. This INIT-ACK chunk establishes the association on the client side, and the alternative handshake is finished.

If the server cannot validate the alternative cookie, for whatever reason, it should generate a new alternative cookie and respond with an ERROR chunk, including this new cookie.

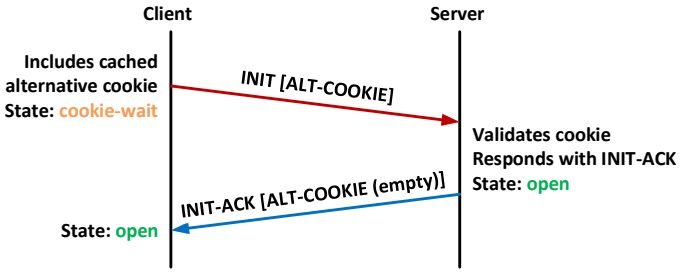


Fig. 5: Alternative handshake using a cached cookie

The client caches the alternative cookie for future associations to the server. To establish new associations, as shown in Figure 5, the client simply includes the previously received alternative cookie in the INIT chunk which instantly opens the association. This method allows an SCTP connection setup in a single round-trip without the disadvantage of re-using association specific parameters like the *Initiate Tags* and the *Initial Transmission Sequence Numbers*.

A. Alternative Cookie Parameter

As already mentioned in the introduction, the INIT and INIT-ACK chunks consist of mandatory and optional parameters. To distinguish the optional parameters, SCTP uses a predefined *Type-Length-Value* (TLV) format to encode the parameter types, variable length and payload.

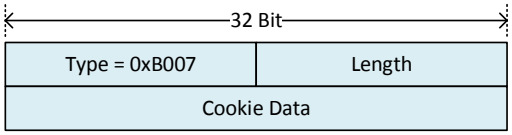


Fig. 6: ALT-COOKIE parameter

The parameter type, represented by a 16-bit field, utilizes the two bits of the highest order to specify the action that must be taken if the processing endpoint does not support the parameter type. The first bit encodes if the endpoint should process any further parameters from the current INIT or INIT-ACK chunk when reading this unknown parameter. If set, an endpoint will stop processing the INIT chunk and report an error. The second bit encodes if an unknown parameter should be reported to the remote endpoint or silently be discarded.

The alternative cookie (ALT-COOKIE) parameter uses this particular TLV structure and is included in the INIT and INIT-ACK chunk for multiple purposes. The ALT-COOKIE parameter type (0xB007) has the highest bit set to one and the second highest bit to zero. This encoding ensures that a server without support for the alternative cookie mechanism will silently discard the ALT-COOKIE parameter and continue processing the SCTP message. The length field contains the size of the parameter in bytes, including the length of the parameter header. An ALT-COOKIE parameter can carry an empty cookie. Therefore, the length value is between 4 and 65535 bytes.

B. Alternative Cookie Calculation

A fundamental requirement in the design process for the alternative cookie was to be more lightweight than the regular state cookie, as already explained in the introduction. This covers the cookie size and its computation time for the server while still offering the same protection level as the regular state cookie. In contrast to the regular state cookie, the only purpose of the alternative cookie is the validation of the client's ownership of the source IP address and, in contrast to the regular state cookie, not to create the TCB on the server side.

The generation and validation of the cookie is in the responsibility of the server. We suggest that an implementation should use an appropriate cryptographic hashing mechanism to ensure the integrity of the cookie. Since the client only reflects

the cookie, the server can embed any desired information in the cookie but should keep it as small as possible to mitigate the risk of amplification attacks. Our implementation uses a SIPHASH [11] based keyed-hash message authentication algorithm, which is also used for the TCP Fast Open cookie in the FreeBSD kernel stack, to generate the cookie from the client's IP address. This method binds the alternative cookie to a specific IP address and, therefore, a multihomed client has to use the same path it received the alternative cookie on for future associations. The lifetime of the alternative cookie can be limited by changing the secret key of the cryptographic hash function.

C. Cookieless Handshake

Both handshake methods, the regular and the alternative, aim to prevent amplification attacks and resource exhaustion by malicious peers. But in certain use cases this protection is not necessary, for example if the SCTP communication is transmitted within a protected environment. The usage of SCTP for WebRTC Data Channels represents a typical case for a protected environment due to DTLS encapsulation of the SCTP communication [12]. In this case, the client and the server already perform a DTLS handshake before the first SCTP message is transmitted within the DTLS tunnel. For those scenarios, the alternative handshake method allows an SCTP server to waive the cookie exchange and open the association upon receiving the INIT chunk with an empty ALT-COOKIE parameter.

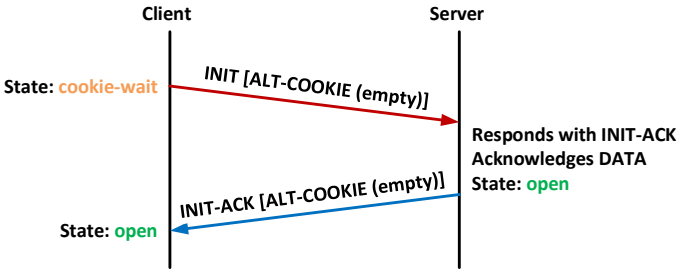


Fig. 7: Cookieless Handshake

Instead of responding with an ERROR chunk, the server acknowledges the successfully established association by sending an INIT-ACK chunk which includes an empty ALT-COOKIE parameter, similar to the last two steps of the alternative handshake with a cookie exchange. This method saves a full round-trip even if both peers have never been in contact before and requires no changes at the client side.

D. Zero-RTT connection setup

In addition to the alternative cookie parameter, our approach introduces an alternative data (ALT-DATA) parameter and an alternative selective acknowledgment (ALT-SACK) parameter. These parameters allow the client to embed application data in the INIT chunk, resulting in a zero RTT connection setup. The ALT-DATA parameter simply acts as a container for regular DATA chunks, see Figure 8. The local endpoint embeds one or more regular DATA chunks in the parameter's value field. The sender has to be aware of not exceeding the path

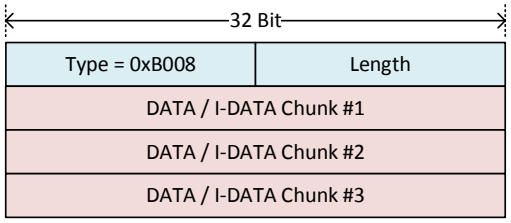


Fig. 8: ALT-DATA parameter

MTU to avoid IP fragmentation when embedding application data.

When the server receives and accepts DATA chunks from the INIT chunk's alternative data parameter, it acknowledges them by an alternative selective acknowledgment parameter (ALT-SACK) which is included in the responding INIT-ACK chunk. This indicates the successful usage of the alternative data parameter to the client.

If the server receives a DATA chunk with an invalid stream number, it drops the DATA chunk according to the procedure specified by RFC4960 and reports it to the client using an ERROR chunk with the *Invalid Stream Identifier* cause. Another case is a server not opening the association and requesting an alternative cookie exchange by sending an ERROR chunk. In this case the application data is also lost and should be retransmitted in the next INIT chunk.

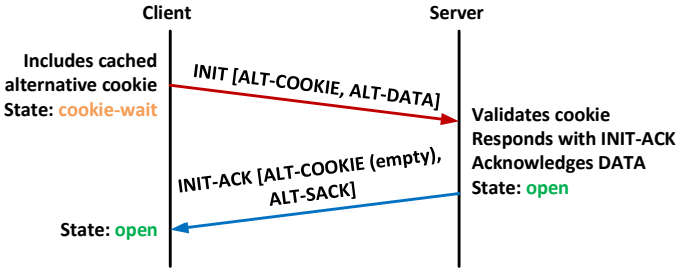


Fig. 9: Alternative Zero-RTT connection setup

A server not supporting data transmission at all via the alternative data parameter will silently discard it and, therefore, not include an ALT-SACK parameter in the responding INIT-ACK chunk. The client retransmits the previously included DATA chunks using the regular way.

E. Fallback Mechanism

A major requirement for the alternative handshake is the seamless backwards compatibility to endpoints not supporting the alternative handshake method, and if the fallback mechanism is used, it should not be unfavorable compared to the regular handshake. As already explained in the previous sections, we use TLV parameters in the INIT chunk to achieve a maximum of compatibility with peers not supporting our new extension. If a server without support for the alternative handshake procedure receives an INIT chunk with an ALT-COOKIE parameter, the server will silently discard the unknown parameter and respond with a regular INIT-ACK chunk not containing the ALT-COOKIE parameter. The client

continues with a regular handshake and reflects the received state cookie. When the client connects to the server for the first time, without having a cached cookie, the alternative cookie parameter has no payload and has a length of four bytes.

The behavior of silently ignoring the unsupported parameter is also used for the alternative data parameter. When the initiating peer using the alternative handshake procedure includes application data in the INIT chunk, and the server does not respond with an INIT-ACK chunk with the ALT-COOKIE parameter set and an included ALT-SACK parameter, the application data is lost and has to be retransmitted using regular DATA chunks. In contrast to wasting four bytes by sending an unused ALT-COOKIE parameter, including application data in an INIT chunk should be considered thoroughly because the performance impact may be much higher. In the worst case scenario where the client and the server support the alternative handshake but only the client supports the alternative data parameter, the client will transmit application data three times before it is accepted by the server. The server responds to the first INIT chunk with an ERROR chunk and requests the reflection of the alternative cookie, while dropping the included DATA chunks. The client retransmits the INIT chunk, now including the alternative cookie which opens the association on the server side. The server ignores the alternative data parameter, and the client has to retransmit the application data a third time using the regular DATA chunk.

F. Multihoming

A key feature of SCTP is multihoming which allows the usage of multiple IP addresses for a single association, either for automatic failover or load sharing [13]. During the handshake, both endpoints exchange their IP address candidates by including them in the INIT and INIT-ACK chunk, respectively. After the association has been established via the primary path, both peers probe the remote address candidates by sending HEARTBEAT chunks. An endpoint responds to a HEARTBEAT chunk by sending a HEARBEAT-ACK chunk which reflects the *Sender-Specific Heartbeat Info* from the HEARTBEAT chunk.

We have modified this mechanism for the alternative handshake since it leads to problems in certain cases. When a client initiates the association by sending an INIT chunk with the alternative handshake method and the server accepts this INIT chunk, either by successfully validating the cookie or by waiving the cookie exchange, the INIT chunk instantly opens the association on the server side and the server responds with an INIT-ACK chunk. Additionally, the server will probe all the client's IP address candidates by sending HEARTBEAT chunks to the particular IP addresses. If the client receives the server's HEARTBEAT chunk before the INIT-ACK chunk, which may happen if the secondary path is faster than the primary or the HEARTBEAT chunk is processed before the INIT chunk, the client cannot match the HEARTBEAT chunk with an existing association and will respond with an ABORT chunk which terminates the association on the server side. To avoid this behavior, the server will not send HEARTBEAT chunks directly after the association has been established. Instead the server waits until receiving the first additional SCTP message from the client on the new association before sending HEARTBEAT chunks to ensure that the association has successfully been established on the client side.

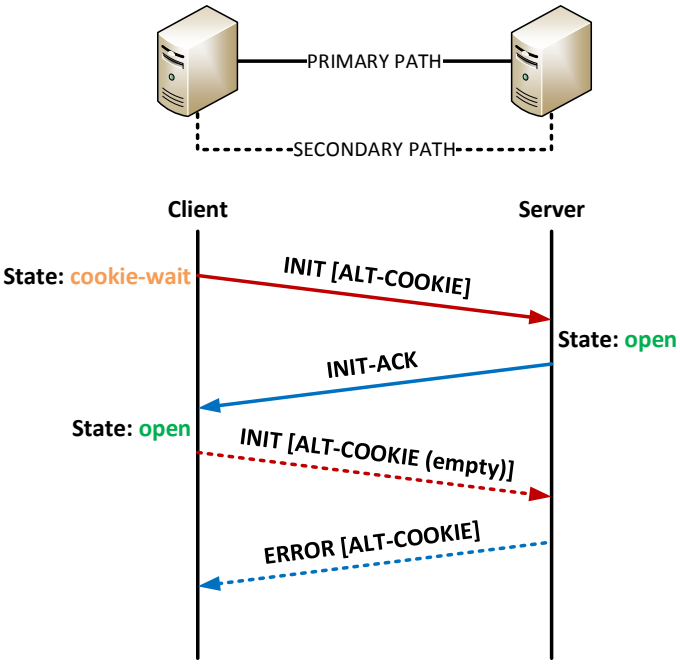


Fig. 10: Alternative cookie exchange on a secondary path

Another change affects the alternative cookie. The alternative cookie only validates the source address of the client from which the INIT chunk has been sent and not the additional address candidates listed in the INIT chunk. Especially mobile devices, like smartphones, often switch between cellular and wifi networks. When establishing a connection to the server, the smartphone uses the wifi network as its primary path and includes its cellular IP address in the INIT chunk as an additional address candidate. When the connection has successfully been established using the primary path, the client has an alternative cookie which validates the ownership of the wifi path but is not valid for a future connection using the cellular path.

To overcome this limitation, the client sends INIT chunks to the server via the additional paths after the association has been established, as shown in Figure 10. The server generates an alternative cookie for the client's source address and responds with an ERROR chunk via the alternative path. The client caches the cookie for future associations on the alternative path.

G. Initialization Collision

Initialization collision happens if both peers initiate an SCTP association by sending an INIT chunk simultaneously and use the same address/port combination. The collision handling is an important feature, especially for the WebRTC use-case where both peers take the active part and initiate the SCTP connection simultaneously, as defined in the corresponding IETF draft [14]. The regular handshake has techniques to detect and handle collision cases, but not all of these techniques can be applied to the alternative handshake since the state cookie is lacking.

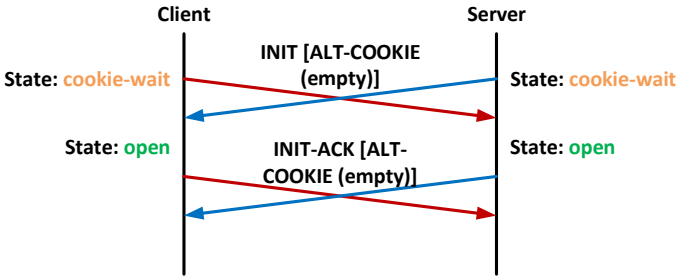


Fig. 11: Initialization collision scenario

When an endpoint receives and accepts an INIT chunk using the alternative handshake procedure from a remote peer while being in a state of waiting for a response for its own INIT chunk from the same address/port combination, as shown in Figure 11, the local endpoint opens the association and responds with an INIT-ACK chunk, following the alternative handshake procedure. But, in contrast to the non-collision case, the local endpoint includes the same parameter as in the previously sent INIT chunk. This especially affects the *initiate tag*. The remote peer receives the INIT-ACK chunk which is carrying the same *initiate tag* parameter as the previously received INIT chunk. This indicates that the local peer has successfully handled the collision case and migrated the colliding connection attempts into a single association.

IV. Extension Compatibility

It is not sufficient to design the alternative handshake mechanism only to match the RFC 4960 since there are additional extensions relying on the traditional handshake mechanism.

A. User Message Interleaving

The *User Message Interleaving* (I-DATA) [15] extension is a mandatory part of the WebRTC Data Channel specification [5], solving SCTP's sender side head-of-line blocking issues when sending a large user message which blocks all other streams. During the regular handshake, both peers announce support for the I-DATA extension in the *Supported Extensions Parameter* as defined in RFC 5061 [16]. If the I-DATA extension has been negotiated, which is the case if both peers have announced to support it, both peers must use the I-DATA chunk instead of the DATA chunk. Using the DATA chunk during an association when the I-DATA extension has been negotiated results in an error. The alternative handshake, including the cookieless variant, does not affect the I-DATA negotiation process but if the I-DATA announcing client wants to include application data within the INIT chunk by using the ALT-DATA parameter, the support for this extension has not been negotiated at this point of time. We considered several ways to solve this issue and concluded that the client should proactively use the I-DATA chunk instead of the DATA chunk when including data using the alternative data parameter in the INIT chunk.

When the alternative handshake has successfully finished, the client handles the application data within the alternative data parameter as lost if the I-DATA extension support has not been negotiated and automatically retransmits the application data using a regular DATA chunk.

A server without support for the I-DATA extension will silently discard the I-DATA chunks carried by the alternative data parameter and wait for the client to retransmit the application data using a regular DATA chunk.

B. Authenticated Chunks

SCTP's 32-bit verification tags protect the association against a blind attacker but not against a Man-in-the-middle attack where the attacker can easily inject SCTP messages with the correct verification tags. The *Authenticated Chunks* [17] extension solves this issue by allowing the endpoints to authenticate specific peer chunks to verify that the chunks are sent by the remote endpoint and not from a Man-in-the-middle attacker. A sender bundles an authentication chunk with the chunk types which should be authenticated. The authentication chunk contains an HMAC which can be used by the receiver to validate the chunks bundled within the received SCTP message.

The support of this extension and the association specific parameters, like the list of authenticated chunks and the HMAC algorithm, are negotiated during the INIT and INIT-ACK chunk exchange. This procedure is compatible with the alternative handshake using the alternative cookie. However, this extension is not compatible with the usage of the ALT-DATA parameter to include DATA chunks in the INIT chunk because the sending endpoint has no knowledge of the receivers capabilities.

V. Implementation

We have successfully implemented and tested the alternative handshake mechanism for the FreeBSD kernel stack and the SCTP userland implementation (usrsctp) [18]. The userland implementation is supported on all widely used operating systems, including FreeBSD, Linux, macOS and Windows. Additionally, the userland implementation is used for WebRTC Data-Channels in several major browsers, including Google Chrome, Mozilla Firefox, Opera and Apple's Safari.

A. API extension

The alternative handshake mechanism is controllable by using the *setsockopt()* and *sysctl()* function calls. The support for the alternative handshake in general is controllable system wide by sysctl variables. System administrators can choose between only allowing the regular handshake (0), the alternative one (2) or both of them (1).

Applications, which want to use the alternative handshake for future associations, use the *setsockopt()* function call to control the functionality per socket. By setting the *SCTP_ALT_HANDSHAKE* socket option, the alternative handshake procedure is activated. The available option values are the same as for the system wide *sysctl* settings. After the association has been established, the application may use the same options for the *getsockopt()* function call to determine if the association has been established using the alternative or the regular handshake.

If the server wants to allow the cookieless alternative handshake, it sets the *SCTP_EMPTY_ALT_COOKIE* socket option on a listening socket. This socket option controls if

```
int sockfd = socket(...);
struct sctp_assoc_value av = { .assoc_id = 0, .assoc_value = 1 };
setsockopt(sockfd, IPPROTO_SCTP, SCTP_ALT_HANDSHAKE, &av, sizeof(av));
setsockopt(sockfd, IPPROTO_SCTP, SCTP_INIT_ALT_DATA, &av, sizeof(av));
char payload[13] = "Hello Irene!";
sendto(sockfd, payload, strlen(payload), ...);
```

Listing 1: Sample code for a client using the alternative handshake with zero-RTT connection setup

a server accepts an empty ALT-COOKIE parameter in the INIT chunk as described in the previous section. For existing associations, it allows to query whether the empty cookie method has been used or not on a particular association, e.g. for statistical usage.

If the network socket is configured to support the alternative handshake, the application may bundle application data within the INIT or INIT-ACK chunk using the ALT-DATA parameter. The application can enable this feature by using the *SCTP_INIT_ALT_DATA* option for the *setsockopt()* function call. Since the SCTP network stack needs the application payload before sending the initiate message, application developers cannot use the typical function sequence of *connect()* and *send()*. Calling the *sendto()* function initiates an implicit connection setup and includes the given payload data in the INIT chunk.

Listing 1 shows a simplified SCTP client example using the alternative handshake procedure with application data included in the ALT-DATA parameter of the INIT chunk by using an implicit connection setup.

VI. Measurements and Evaluation

To evaluate the performance of our alternative handshake procedure and its implementation, we created a client-server scenario as shown in Figure 12. Both nodes have the identical hard- and software configuration: PC Engines APU2 boards with mSATA solid state discs and two operating systems installed. We have chosen this hardware configuration by intention to evaluate CPU effects caused by the low performance and energy optimized processor. In addition to FreeBSD HEAD with release type kernel and disabled debugging options (GENERIC-NODEBUG), we used Ubuntu 17.10.

Fig. 12: Testbed for INIT chunk flooding

A. INIT flooding

Although SCTP is robust against INIT-flooding attacks, similar to TCP SYN-flooding, however, generating the state cookie consumes a large amount of CPU load for the server side, and the INIT-ACK chunk is significantly larger than the INIT chunk. In a first step, we evaluated the performance of the INIT chunk handling on the server side for different scenarios where a server, running FreeBSD HEAD and Ubuntu 17.10, is flooded with INIT chunks. To send a large amount of SCTP messages containing an INIT chunk while having the greatest possible control over the senders behavior, we developed a benchmark tool using the netmap fast packet I/O framework [19], allowing us to send and receive network packets with a variable rate up to wire speed. To eliminate side effects, we disabled the ethernet flow control on all nodes. The client, running the netmap benchmarking tool, sends SCTP messages containing an INIT chunk with different rates to the server for a fixed timespan of 60 seconds and measures several values during the test run, including the packet-rate, bandwidth and average packet size in both directions. The benchmark tool provides several options to modify the INIT chunks, which includes extensions, address candidates and the number of parameters. We have configured the benchmark tool to flood the server with INIT chunks which announce support for all officially specified extensions and additionally our alternative handshake extension.

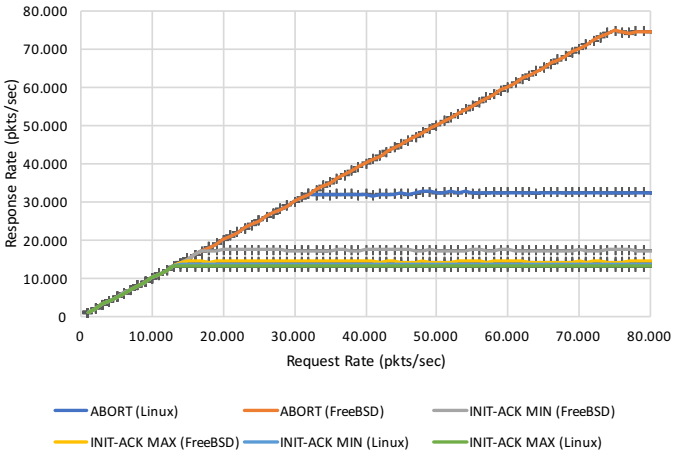


Fig. 13: Comparison of Linux and FreeBSD INIT-ACK rates

In the first test scenario, the client sends small SCTP messages of 112 bytes, containing an INIT chunk, to the server which does not have a listening service on the particular IP/port combination. The server responds with an ABORT chunk which is a cheap operation for the server, its generation consumes only a small amount of resources and its size is only 16 bytes. Therefore, this is our baseline scenario with respect to the responding packet rate and resource consumption on the server side. As shown in Figure 13, we measured an ABORT rate of 74 kpps from the FreeBSD server and an ABORT rate of 32 kpps from the Ubuntu server.

In our second scenario, the server has a listening socket bound to the specific IP/port combination and responds to INIT chunks with an INIT-ACK chunk. As already mentioned

in the previous section, the INIT-ACK chunk is significantly larger than the INIT chunk and its generation is more resource consuming. In contrast to Linux, FreeBSD announces all officially specified extensions in the INIT-ACK chunk by default, even if they are not requested by the INIT chunk. To have a more comparable result, we measured the INIT-ACK rate on both machines with two different settings. First with all extensions enabled, labeled as *INIT-ACK MAX* in Figure 13 and additionally with all extensions disabled, labeled as *INIT-ACK MIN*. In contrast to the ABORT rate, the INIT-ACK rates of Ubuntu and FreeBSD are on a similar level. The INIT-ACK rates for both systems range between 13 kpps for an INIT-ACK chunk with all extensions from the Ubuntu machine until 18 kpps INIT-ACK chunks without any extensions from the FreeBSD machine.

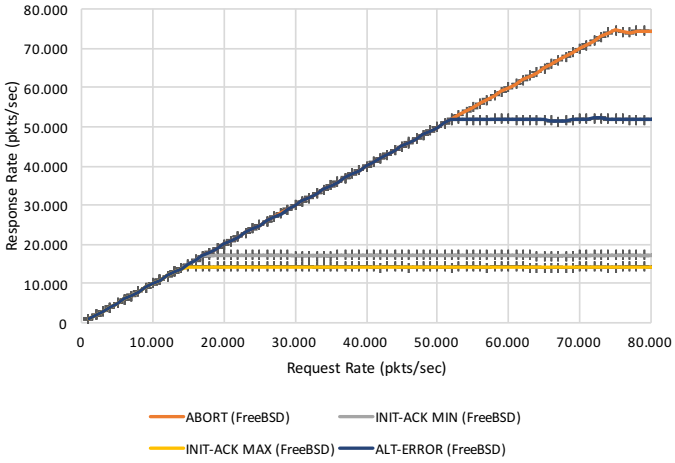


Fig. 14: FreeBSD's response rate to INIT chunks

In a third step we used the alternative handshake method where the server responds with an ERROR chunk, bundled with the alternative cookie. The server is able to respond with about 52k ERROR chunks per second which carry the alternative cookie, shown in Figure 14. In addition, the responses are much smaller compared to the regular INIT-ACK chunk. In our scenario, the regular INIT-ACK chunk, with extensions, has a length of 416 bytes whereas the alternative one has a length of only 40 bytes. While the alternative response is only 4 bytes larger than the initiating request, the regular response is more than ten times larger than the SCTP message containing the INIT chunk. This shows another advantage of the alternative handshake, it successfully prevents byte amplification attacks.

B. Time to first byte

A common use case of SCTP, since it is reliable and message oriented, is the transmission of small messages in a request-response manner. Typically the client sends a small request, the server answers with a response and closes the connection afterwards. This traffic pattern is common for signaling services and measurement grids. We developed a client and a server to evaluate the performance improvements of the alternative handshake procedure over the regular handshake.

The client establishes a connection to the server and sends a small request of less than 100 bytes payload. The server

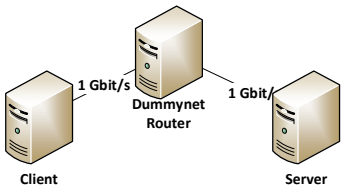


Fig. 15: Testbed for signaling traffic

also responds with a small message of less than 100 bytes and closes the connection afterwards. We varied the link delay by using the dummynet [20] network emulation tool running on a router between the server and the client, this scenario is shown in Figure 15. The link speed has been set to 2 Mbit/s, and we varied the link delay.

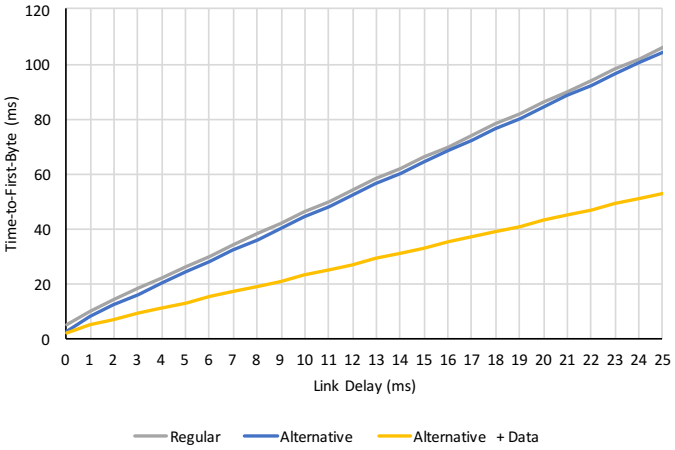


Fig. 16: Timespan between the connection initialization and receiving the first bytes on the client side

Figure 16 shows the results for three different handshake types. We measured the timespan between initiating the association and the arrival of the server's response on the client side. The regular handshake and the alternative handshake show a nearly identical performance, the alternative handshake is slightly faster because of the smaller cookie size. This matches our expectations since both peers are not affected by CPU limitations and both handshake variants take the same amount of round-trips. When the client uses a previously cached cookie, the connection setup time is reduced by one round-trip. Since both DATA chunks, containing the request and the response, fit in a single SCTP message, the timespan until the server's response arrives at the client is reduced by one half when using a previously cached cookie.

C. Compatibility and Security

We have tested the backwards compatibility of our new extension with multiple SCTP implementations without support for the extension to ensure its deployability. This includes the implementations of FreeBSD, Linux, Solaris and the userland stack. All of them successfully ignore the alternative parameter in the INIT chunk and continue with the regular handshake by sending an INIT-ACK chunk. Thus, and since introducing new parameters is defined in the official RFC4960, we do not expect any compatibility problems in deployment.

Our measurements and evaluation demonstrates that an attacker is able to exhaust the server's CPU resources by sending a large number of INIT chunks and, since the generated INIT-ACK chunk is mostly larger than the corresponding INIT chunk, may also be used for a byte amplification attack. This insight is not new and has already been treated by RFC5062 [6] in the year 2007, where this kind of attack is characterized as hard to avoid. RFC5062 suggests to use the PAD parameter [21] to artificially enlarge the initiating message and, therefore, prevent this attack pattern.

The QUIC protocol makes use of this method, the initiating QUIC message must at least have a length of 1200 octets. Our implementation allows the server to waive the fallback mechanism and only support the alternative handshake. When configured to do so, a server will always respond with an ERROR chunk including an alternative cookie upon receiving an INIT chunk, even if the client has not announced support for the alternative handshake. This is an effective way to prevent amplification attacks but requires both peers to support the alternative handshake.

Before an application developer enables the new handshake features, their possible drawbacks should be considered carefully. While using the alternative cookie parameter should not have any negative impact, as it has a seamless fallback mechanism and offers the same protection as the regular handshake, the inclusion of application data in the INIT chunk may lead to unwanted behavior regarding security and data integrity. The cookieless handshake should only be enabled in protected environments, like WebRTC Data-Channels.

VII. Conclusion and Outlook

This paper introduces an alternative handshake mechanism for the SCTP protocol which reduces the required round-trips for association establishment and offers a lower resource consumption. Our solution provides the same protection against INIT-flooding attacks as the regular handshake procedure and is fully backwards compatible to peers not supporting the new extension. It can also prevent byte amplification attacks in case the server waives the backwards compatibility by only accepting handshakes using the alternative mode. In certain scenarios, like the usage of SCTP for WebRTC Data-Channels, our new zero RTT connection setup capability gives a significant performance improvement, compared to the regular handshake. We have implemented the new handshake mechanism for the FreeBSD kernel stack and the widely used userland stack and evaluated its impact in several test scenarios with both implementations regarding robustness against INIT-flooding attacks and performance improvements. Our future work will focus on improving the alternative handshake and its implementation for the WebRTC Data-Channel use-case. We are also working on an IETF draft and will publish our implementation.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT) and the German Research Foundation (Deutsche Forschungsgemeinschaft). The views expressed are solely those of the authors.

References

[1] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413 (Experimental), December 2014.

[2] Eric Rescorla. The transport layer security (tls) protocol version 1.3. Internet-Draft draft-ietf-tls-tls13-28, IETF Secretariat, July 2017. http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-28.txt.

[3] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-10, Internet Engineering Task Force, March 2018. Work in Progress.

[4] R. Stewart (Ed.). Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[5] Randell Jesup, Salvatore Loreto, and Michael Tuexen. Webrtc data channels. Internet-Draft draft-ietf-rtcweb-data-channel-13, IETF Secretariat, January 2015. http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-data-channel-13.txt.

[6] R. Stewart, M. Tuexen, and G. Camarillo. Security Attacks Found Against the Stream Control Transmission Protocol (SCTP) and Current Countermeasures. RFC 5062 (Informational), September 2007.

[7] E. P. Rathgeb, C. Hohendorf, and M. Nordhoff. On the robustness of sctp against dos attacks. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, 2008.

[8] I. Joe and L. Kant. Sctp with an improved cookie mechanism for wireless networks through modeling and simulation. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, 2003.

[9] I. Joe. Sctp with an improved cookie mechanism for mobile ad-hoc networks. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 2003.

[10] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012. Updated by RFCs 7507, 7905.

[11] SipHash: a fast short-input PRF. https://131002.net/siphash/. [Online; accessed 24-November-2017].

[12] M. Tuexen, R. Stewart, R. Jesup, and S. Loreto. Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets. RFC 8261 (Proposed Standard), November 2017.

[13] Professor Paul D. Amer, Martin Becke, Thomas Dreibholz, Nasif Ekiz, Jana Iyengar, Preethi Natarajan, Randall R. Stewart, and Michael Txen. Load Sharing for the Stream Control Transmission Protocol (SCTP). Internet-Draft draft-tuexen-tsvwg-sctp-multipath-15, Internet Engineering Task Force, January 2018. Work in Progress.

[14] Christer Holmberg, Roman Shpount, Salvatore Loreto, and Gonzalo Camarillo. Session description protocol (sdp) offer/answer procedures for stream control transmission protocol (sctp) over datagram transport layer security (dtls) transport. Internet-Draft draft-ietf-mmusic-sctp-sdp-26, IETF Secretariat, April 2017. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-sctp-sdp-26.txt.

[15] R. Stewart, M. Tuexen, S. Loreto, and R. Seggelmann. Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol. RFC 8260 (Proposed Standard), November 2017.

[16] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. RFC 5061 (Proposed Standard), September 2007.

[17] M. Tuexen, R. Stewart, P. Lei, and E. Rescorla. Authenticated Chunks for the Stream Control Transmission Protocol (SCTP). RFC 4895 (Proposed Standard), August 2007.

[18] usrsctp - a portable SCTP userland stack. *Available at https://github.com/sctplab/usrsctp*, 2018.

[19] Luigi Rizzo. Netmap: A novel framework for fast packet i/o. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[20] The dummynet project. http://info.iet.unipi.it/~luigi/dummynet/. [Online; accessed 19-September-2017].

[21] M. Tuexen, R. Stewart, and P. Lei. Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP). RFC 4820 (Proposed Standard), March 2007.

A Framework for Evaluating Caching Policies in A Hierarchical Network of Caches

Eman Ramadan, Pariya Babaie, Zhi-Li Zhang
Department of Computer Science and Engineering
University of Minnesota, Twin Cities
Email: eman, babai008, zhzhang@cs.umn.edu

***Abstract*—Much attention of the research community has focused on performance analysis of cache networks under various caching policies. However, the issue of how to evaluate and compare caching policies for cache networks has not been adequately addressed. In this paper, we propose a novel and general framework for evaluating caching policies in a hierarchical network of caches. We introduce the notion of a hit probability/rate matrix, and employ a generalized notion of majorization as the basic tool for evaluating caching policies for various performance metrics. We discuss how the framework can be applied to existing caching policies, and conduct extensive simulation-based evaluation to demonstrate the utility and accuracy of our framework.**

I. Introduction

The emergence of information-centric network (ICN) architectures [1], [2], [3], [4], [5] (see [6] for a survey of ICN architectures) has attracted a flurry of renewed research interest in caching policies and their performance analysis. Classical caching policies such as FIFO, LRU, or LFU – which specify what object should be evicted when the cache is full – have been widely used in computer systems. It is notoriously difficult to exactly analyze the performance of these caching policies, instead one has to resort to approximation methods with various assumptions [7], [8], [9]. These policies can be viewed as organizing the cached objects in an ordered list for replacement. More recently, timer-based caching policies have gained particular attention (see, *e.g.*, [10], [11], [12], [13]) – where each object is associated with a time-to-live (TTL) timer, and is evicted when the timer expires. The interest in timer-based caching policies is due to the fact that objects within a cache can be viewed independently, and their hitting probabilities can be analyzed separately. As an analytical aid, TTL-cache can provide a good avenue to approximate classical list-based caching policies [14], *e.g.*, in terms of characteristic times [7], [15], [16].

One important feature ICNs offer is a distributed network of caches, namely, a *cache network*, which poses additional challenges both in terms of practical cache management issues and performance analysis. For example, when an object is evicted from one cache, should it simply be discarded, or should it be inserted into the next cache (*i.e.*, as a *new arrival* to this next cache) along the path to the origin server? When an object is returned from an upstream cache (or the origin server) back down along the request path, should it be cached along the way (*e.g.*, as advocated by [1], [2]), selectively or probablistically at some caches (*e.g.*, leave-copy-down or leave-copy-probabilistically [7]), or only at the edge (*e.g.*, as advocated by [4])? Should caches in a network be operated independently, in a cooperative fashion [17], or managed *globally* with a coherent view [18]?

Moreover, no single caching policy is likely to perform well for *all* user request patterns. For example, under the ***I**ndependence **R**eference **M**odel (IRM)*, static caching is shown to be optimal [19] if the object popularity distribution is known *a priori*. However, it is shown in [20] that static caching is no longer optimal when the interarrival distribution of object requests has a decreasing hazard rate (*e.g.*, when the interarrivals of object requests are Pareto-distributed). In this case, instead of always caching the most popular objects as in the case of static caching, the optimal policy is to cache each object with a probability less than 1. This means that objects do not have to be always in the cache, leaving space for more objects to be cached. In addition, static caching cannot cope with changes in user request patterns, *e.g.*, a flash-crowd.

From a theoretical standpoint, performance analysis of a network of caches is significantly more difficult: consider the simple case of a line of caches where each cache employs its own cache replacement policy (*e.g.*, LRU) independently; assuming that object request streams at the first cache (the *edge* cache) are independent, the request arrival streams at the upstream caches are no longer independent – they are generated by cache misses from the downstream caches. Such coupling of the caches in tandem network is what makes the analysis of cache networks a challenging task. Approximation results for cache networks of specific topologies (*e.g.*, a line or star network) have been obtained for LRU caches (see, *e.g.*, [7], [15], [16]); exact and approximation results for a general network of TTL-caches have been developed recently under either the renewal arrival processes or Markov arrival processes (MAP) [7], [12], [13]. In our previous work [18], we have shown that when caches in a network are operated independently (with their own cache replacement policy such as LRU), the utilization of intermediate caches can be extremely poor, due to the "thrashing" problem caused by the (non-independent) *filtered* (cache miss) arrival processes at intermediate servers. To address this problem, we proposed the innovative notion of *"BIG" cache abstraction* [18] by viewing a line of caches from an edge server along the path to the origin

ISBN 978-3-903176-08-9

server as a single "BIG" cache, and argued that it affords the added benefits of simplifying the analysis of a line of caches.

Unlike a single cache where the performance of various caching policies can be directly compared, evaluating and comparing the (relative) performance of caching policies for a network of caches are no longer straightforward. From a user's perspective, hit probabilities at individual caches are immaterial; what matters is the latency he/she experiences. On the other hand, a cache network provider is more concerned with the efficient utilization of *all* cache capacities in the network; whereas from the standpoint of a content provider, the utility of a cache network is its ability to decrease the overall load on its origin server(s), and reduce the network bandwidth cost (it also cares about improved content access latencies to its users). Despite much focus on performance analysis of cache networks, this important problem has not received much attention in the research community.

In this paper, we propose a novel and general framework to evaluate and compare caching policies for a network of caches. Consider a collection of content objects served by a network of caches with a fixed set of ingress points (or edge servers) where user requests for content are routed. We assume caches are organized in a hierarchy, from the edge servers to the origin server(s). Given the request streams for the collection of objects at each ingress point (edge server), we introduce the notion of a *hit probability matrix*, which characterizes the hit probability of content objects that are served at different layers of the cache hierarchy. We employ (and define an extended version of) the *majorization* notion as the basic tool to evaluate and compare caching policies for various performance metrics of cache networks in Section II.

Section III provides an overview of the existing works to estimate the hit probability matrix, including their limitations. Then, we provide and evaluate a general simplified approach to estimate the hit probability matrix based on the "BIG" cache abstraction in Section IV. As we have shown in [18], implementing caching policies as a single caching strategy for the virtual "BIG" cache outperforms their implementation at each layer independently. In this paper, we show that our approach to estimate the hit probability matrix achieves the exact result as the "BIG" cache simulation for LRU and q-LRU caching strategies as examples. Thus, this estimation approach can be used to generate the hit matrix for the comparison framework without wasting any time on simulations. Section V shows the accuracy of applying our proposed framework to compare caching policies for different user request patterns. Finally, the paper is concluded in Section VI.

II. Caching Policies Comparison Framework

In this section, we present a general framework to evaluate and compare two caching policies, P and Q, for network-wide performance analysis, where P, Q represent any caching policy such as LRU, static caching, k-LRU, ... etc. We first describe the network model, basic assumptions, and the key notion of hit probability matrix associated with a caching policy (here we assume it is given). Then, we identify the conditions for policy P to *dominate* policy Q by generalizing the notion of majorization defined for vectors to matrices.

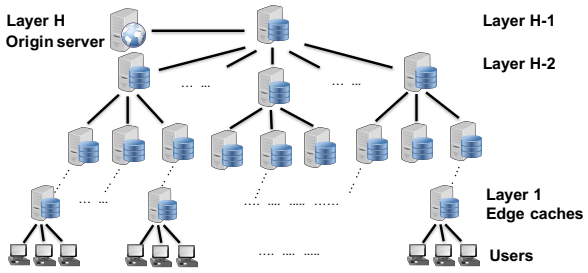


Figure 1. Network Model

A. Network Model and Assumptions

For ease of exposition, we make a simplifying assumption and model the cache network as a hierarchy of cache servers organized in an $(H-1)$ level k-ary tree, as shown in Figure 1, which is commonly used in today's content distribution networks [21], [22]. Cache servers at leaf nodes represent edge servers, which are the closest to users located at layer 1. The root of the tree is connected to the origin server (at layer H, which has a permanent copy of each object). The content population is a collection of N unique objects of unit size, denoted by $\mathcal{O} = \{O_1, O_2, \ldots O_N\}$. The popularity of objects follows a Zipf distribution with parameter α. The access probability of each object is denoted by $a_i = \lambda_i/\lambda$, where λ_i is the request rate of object i, and λ is the aggregate request rate, $\lambda = \sum_i \lambda_i$. The access probability a_i is proportional to $\frac{1}{i^\alpha}$ for $\alpha > 0$, and $\sum_{i=1}^{N} a_i = 1$. Without loss of generality, we assume that $a_1 \geq a_2 \geq \ldots \geq a_N$, namely, O_1 is the most popular object, O_2 is the second most popular object, and so forth. Object requests arrive randomly at one of the edge servers. When a request is received at a server, which does not have the object, it forwards the request to its parent. This process continues till the request reaches the root, and the origin server eventually if no other server on the path has a copy. Each request experiences a latency depending on the layer it is served from. First, we consider comparing the caching policies P and Q for a tandem cache network (a line of caches) starting from one edge cache C_e at layer 1, till the origin server at layer H. The concepts and notations introduced below can be generalized to a cache network of any arbitrary topology, where we construct a request routing tree/graph formed by the request forwarding paths from each edge server towards an origin server as illustrated at the end of this section.

B. Tandem Cache Network

We define L_j to denote the latency experienced by an object served from a cache server at layer j. We assume $L_1 < L_2 < \ldots < L_H$ and $L_j = L_{j-1} + \Delta L_{j-1},\ 2 \leq j \leq H$. For a caching policy P, $a_i p_{ij}$ represents the hit probability of object O_i at layer j cache, where $1 \leq j \leq H$, p_{ij} is the percentage of requests for O_i served from layer j cache, and a_i is the access probability of object O_i.

The set of values $\{a_i p_{ij}\}, 1 \leq i \leq N, 1 \leq j \leq H$ can be compactly represented using an $N \times H$ matrix, and by abuse of notation, we denote it as "P".

$$P_{N\times H} = \begin{bmatrix} a_1p_{11} & a_1p_{12} & a_1p_{13} & \dots & a_1p_{1H} \\ a_2p_{21} & a_2p_{22} & a_2p_{23} & \dots & a_2p_{2H} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_Np_{N1} & a_Np_{N2} & a_Np_{N3} & \dots & a_Np_{NH} \end{bmatrix}$$

Majorization of Hit Probability Matrices. We employ the notion of *majorization* as the basic tool to evaluate and compare caching policies for a cache network. The standard notion of majorization is defined for vectors. We generalize it to a (hit probability) matrix as follows. Given two caching policies P and Q, with hit probability matrices represented by $P = [a_i p_{ij}]$ and $Q = [a_i q_{ij}]$ respectively. Without loss of generality, we assume both policies are "sensible" at layer 1 cache – namely, the first column is decreasing in value. As caching policies react based on the received requests, thus popular objects are cached more often than others, which is also confirmed by the simulation results shown in [18] for different caching policies. Therefore, we assume the following holds:

$$\begin{aligned} p_{11} \geq p_{21} \geq p_{31} \geq \cdots \geq p_{N1} \\ q_{11} \geq q_{21} \geq q_{31} \geq \cdots \geq q_{N1} \end{aligned} \tag{1}$$

Also, the summation of the hit probabilities of object O_i for all H layers equals 1 under both policies, *i.e.*, a request for object O_i has to be served from one of the H layers.

$$\sum_{j=1}^{H} p_{ij} = \sum_{j=1}^{H} q_{ij} = 1, \quad \forall \ 1 \leq i \leq N \tag{2}$$

Given the above conditions, we say P *majorizes* Q, denoted by $P \succ Q$, if the following criterion holds:

The summation of the hit probabilities in the top-left (k, h) sub-matrix of P is equal or larger than that of Q for all values of k, h, where $1 \leq k \leq N, 1 \leq h \leq H$, (*i.e.*, policy P utilizes the first h cache layers to serve the top k objects better than policy Q).

$$\sum_{i=1}^{k}\sum_{j=1}^{h} a_i p_{ij} \geq \sum_{i=1}^{k}\sum_{j=1}^{h} a_i q_{ij} \tag{3}$$

Thus, we say caching policy P **dominates** Q if and only if $P \succ Q$ (*i.e.*, P majorizes Q).

Comparing Overall Performance. Having two general caching policies P, Q, where P dominates Q (from the perspective of the edge server), we show that P outperforms Q in terms of both the overall latency (as seen by the end user), and the load of the origin server.

*1) **Expected Overall Latency**:*

Theorem 1: For a hierarchy of caches of H layers, if caching policy P dominates caching policy Q, the overall expected latency for P is less than or equal to that of Q.

Proof: Expected latency under caching policy P (OL_P):

$$OL_P = \sum_{i=1}^{N}(a_i \sum_{j=1}^{H}(p_{ij} L_j)) \tag{4}$$

Since, $L_j = L_H - \sum_{h=j}^{H-1} \Delta L_h, \ 1 \leq j \leq H-1$

By substituting for L_j in "(4)", and rearranging the summation indices:

$$OL_P = \sum_{i=1}^{N}(a_i \sum_{j=1}^{H}(p_{ij} L_H)) - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}(a_i \sum_{j=1}^{h} p_{ij}))$$

From "(2)" and $\sum_{i=1}^{N} a_i = 1$:

$$OL_P = L_H - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}\sum_{j=1}^{h} a_i p_{ij})$$

From "(3)":

$$OL_P \leq L_H - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}\sum_{j=1}^{h} a_i q_{ij})$$

$$OL_P \leq OL_Q$$

Intuitively, caching policy P dominates Q means that under policy P, the top k most popular objects are likely to be placed in the first h layer caches than under policy Q, for $1 \leq k \leq N$, $1 \leq h \leq H$. Given that $L_1 < L_2 < \cdots < L_H$, we would expect that P outperforms Q in terms of expected latency. ■

*2) **Origin Server Load**:*

Theorem 2: For a hierarchy of caches of H layers, if caching policy P dominates caching policy Q, the origin server load under policy P is less than or equal to that of policy Q.

Proof: Origin server load under caching policy P (S_P):

$$S_P = \sum_{i=1}^{N} a_i p_{iH} \tag{5}$$

From "(2)" and $\sum_{i=1}^{N} a_i = 1$:

$$S_P = 1 - \sum_{i=1}^{N}\sum_{j=1}^{H-1} a_i p_{ij}$$

From "(3)":

$$S_P \leq 1 - \sum_{i=1}^{N}\sum_{j=1}^{H-1} a_i q_{ij}$$

$$S_P \leq S_Q$$

Intuitively, caching policy P dominates Q means that under policy P, the top k most popular objects are likely to be placed in the first h layer caches than under policy Q, for $1 \leq k \leq N$, $1 \leq h \leq H$. Since, the number of requests is directly proportional to the object popularity, we expect more requests to be satisfied form the first h layer caches under policy P. Thus, the origin server load under policy P would be less than or equal to that of Q. ■

C. General Cache Networks

We now discuss how this framework can be utilized for any general cache network. Consider a cache network as shown in Figure 1. Each edge server $C_e \in \mathcal{E}$ (set of all edge servers) is deployed to service content requests from one user populace located close to C_e. The content population of edge C_e is denoted by $\mathcal{O}_e = \{O_1^e, O_2^e, \dots O_{N^e}^e\}$, and the object popularity follows a Zipf distribution with parameter α_e. The access probability of each object is denoted by $a_i^e = \lambda_i^e / \lambda_e$, where λ_i^e is the request rate of object O_i^e, λ^e is the aggregate request rate for the edge server C_e, and $\lambda_e = \sum_i \lambda_i^e$.

A user's request for an object O_i^e is first routed to the edge server C_e closest to her. The request is serviced directly by

C_e if it has O_i^e cached; otherwise C_e routes the request along a request path $\mathcal{L}^e$ – consisting of a sequence of intermediate cache servers, $C_h^e \in \mathcal{L}^e$ – towards the origin server. If one of the intermediate servers, C_h^e, has O_i^e cached, the request is serviced by C_h^e, and the cached copy is returned along the reverse path back to C_e, which then delivers it to the user. Otherwise, the request is serviced by the origin server. Thus, we consider the hierarchical topology as multiple tandem cache networks, each corresponding to an edge server C_e. The request paths $\mathcal{L}^e$ from multiple edge servers traverse and share the cache resources at the intermediate cache servers C_h.

Given two caching policies P and Q, and their hit probability matrices $P^{(e)}$ and $Q^{(e)}$ respectively from the prospective of each edge cache server C_e. We can then apply the comparison framework detailed earlier to compare the two caching policies P and Q for each edge server C_e. Clearly, if $P^{(e)} \succ Q^{(e)}$ for each edge server C_e, then $P = \sum_e P^{(e)} \succ \sum_e Q^{(e)} = Q$, where $P = \sum_e P^{(e)}$ is the aggregate hit probability matrix over all edge servers, as shown in Section V. The aggregate hit probability matrix can be determined using approaches such as [7], in which the authors extend their proposed method (discussed in Section III) for a general cache network, and use the miss rate of lower layer caches as an estimate of the request arrival rate for the current cache node. Finally, these concepts and notations can be generalized to a cache network of any arbitrary topology, where we construct a request routing tree/graph formed by the request forwarding paths from each edge server towards an origin server. Moreover, the cache network does not have to be symmetric, the tandem cache network from each edge server till an origin server could have a different height, and the missing layers could be replaced by dummy cache nodes with zero capacity. Hence, their hit probability is zero, and our proposed comparison framework would still be applicable.

III. Hit Probability Matrix Estimation using TTL Caches

As mentioned before, the analysis of cache networks is a complex and challenging task. This is because the request arrival streams at the upstream caches are not independent – they are generated by cache misses from the downstream caches. Thus, for each cache in the network, the miss rate of content objects should be calculated, along with splitting the miss streams to the other upstream caches. Moreover, the superposition of the miss streams which form the requests arriving at the intermediate caches needs to be calculated. Therefore, several approaches (*e.g.*, [7], [12], [15], [23]) have been proposed to estimate the requests arrival rate at intermediate caches. Thus, the object hit probability for every cache in the network can be calculated.

The complexity of the analysis of capacity-based caching policies is due to the dynamics of the content objects in the cache. Hence, Che *et al.* [15] proposed the relationship between a capacity-based caching policy (LRU) and a timer-based caching, by defining the characteristic time. Timer-based caching also simplifies the performance analysis of caching systems, as it decouples the dynamics of content objects within a cache, as they can be analyzed independently. In this section, we categorize some of the existing TTL-based approaches according to the type of the approximations they proposed to estimate the request arrival rates at caches, and briefly discuss their limitations.

A Hierarchical Network of LRU Caches with LCE, LCP, and LCD Replication Strategies. The authors in [7] provided a unified approach to analyze the performance of caching policies such as LRU, FIFO, RANDOM, q-LRU, k-LRU, ... etc. for single cache, by extending the decoupling technique introduced in [15]. The authors also analyzed LRU for a two-layered cache network with respect to the following object replication strategies, which define how the object is cached when it is returned from an upstream cache (or the origin server) back down along the request path. *1) Leave-Copy-Everywhere* (LCE): the object is cached at all the downstream caches along the request path. *2) Leave-Copy-Probabilistically* (LCP): the object is cached at the downstream caches along the request path with a probability q. *3) Leave-Copy-Down* (LCD): the object is cached only at the cache preceding to the one where it is currently cached.

For a single cache, the authors considered a temporal locality relation between requests by considering renewal process for request arrival process. However, the key assumption for a hierarchical network of caches is that the request arrival process at any cache in the network is Poisson (IRM request traffic). The existing spatial-correlation and temporal-correlation among requests are ignored as the requests interarrival process is considered Poisson at the intermediate caches. The authors justified this assumption by mentioning that the error gets smaller as the network grows (in terms of the number of branches), hence, the proposed model becomes valid. Finally, the authors proposed another extension for their model to work for a network of caches with any topology.

Equations "6" & "7" define the hit probability at each layer for object m in an LRU-cache with LCE & LCP object replication strategies respectively. The hit probability of object m at cache i is denoted by $p_{hit}(i,m)$. T_C^i is the timer assigned to all objects at cache i which is related to the cache size, and q is the probability to cache objects for LCP. The *average arrival rate* for object m at cache i is calculated by: $\overline{\lambda}_m(i) = \sum_j \overline{\lambda}_m(j)(1 - p_{hit}(j,m))r_{j,i}$, where $r_{j,i}$ is the probability that cache j forwards its miss stream to cache i. We use these equations in Section V to calculate the hit probability matrix, and compare them using our proposed framework.

LCE

$$\begin{aligned} p_{hit}(1,m) &= 1 - e^{-\overline{\lambda}_m(1)T_C^1} \\ p_{hit}(2,m) &\approx 1 - e^{-\overline{\lambda}_m(2)(T_C^2 - T_C^1)} \end{aligned} \tag{6}$$

LCP

$$\begin{aligned} p_{hit}(1,m) &= \frac{q(1 - e^{-\overline{\lambda}_m(1)T_C^1})}{e^{-\overline{\lambda}_m(1)T_C^1} + q(1 - e^{-\overline{\lambda}_m(1)T_C^1})} \\ p_{hit}(2,m) &\approx [p_{hit}(2,m) + q(1 - p_{hit}(2,m))] \\ &\quad (1 - e^{-\overline{\lambda}_m(2)(T_C^2 - T_C^1)} e^{-\overline{\lambda}_m(2)(1-q)T_C^1}) \end{aligned} \tag{7}$$

Approximate Analysis of Hierarchical and General TTL-Cache Networks. Fofack *et al.* [10], [12], [24] focus on analyzing the performance of LRU, FIFO, and RANDOM caching policies using the characteristic time. The timer is set up so that the number of objects in the cache does not exceed the cache size, considering the same size for all objects. The key assumption of this model is considering the requests arriving at any cache in the network to have a renewal process interarrival time. The authors also assume that requests are forwarded in a feed-forward network in contrast with [7]. This approach characterizes three request streams: i) the miss stream of each cache, ii) the splitted miss streams to next layer caches, and iii) the superposition of request streams arriving from different caches along with exogenous request arrival as a renewal process. The main source of error of this approach is considering the superposition of renewal processes as a renewal process, which is a non-renewal in general. Moreover, this approach is computationally very extensive when the size of the cache network grows, thus, it is not scalable. Finally, it has the limitation of assuming the routing of the cache network to be feed-forward.

Exact Analysis of a Hierarchical Network of TTL-Caches. Following the approach in [12], Berger *et al.* [23] propose an exact model for performance evaluation of a hierarchical network of caches. The key contribution of this approach is adopting Markovian Arrival Process (MAP) for request arrival processes, and showing that the miss stream of TTL-based caches is MAP as well. Since the superposition of MAP processes is also MAP, the provided analysis are the exact values for feed-forwarded cache networks with a MAP process as an input for all caches in the network. However, this approach suffers from the computation cost for large networks, as well as [12], and suffers also in case of non feed-forward cache networks.

IV. General Approach for Hit Probability Matrix Estimation

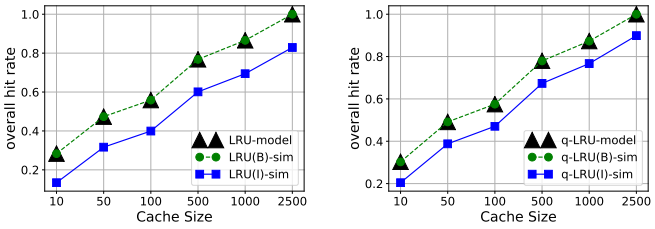
The input to our proposed comparison framework (Section II) is the hit probability matrices of both policies ones wish to compare, which on general is a challenging task to calculate as discussed in Section III. Ether of exact or approximatation calculated values are known for specific network set ups [7], [12], [23], assumptions about the request inter-arrival processes. As mentioned, currently, there exists no *general* methodology for computing, approximating or even bounding the hit probabilities for a *network* of caches within a reasonable computation cost. Even with the simplifying IRM assumption, the request arrival processes to intermediate network caches are *filtered*, and no longer independent. This creates technical difficulty in analyzing a cache network for any arbitrary caching policy under general assumptions of request arrival processes. Thus, in this section for a caching policy P, given the equations to calculate the hit probability for a single cache, we propose a general simplified approach to calculate the hit probability matrix for a tandem cache network, using the notion of "BIG" cache abstraction introduced in [18]. At the end of this section, we extend this approach to be applied for any general cache network.

The main idea of "BIG" cache is to view a group of hierarchical caches as if they are "glued" together to form one virtual "BIG" cache with a storage capacity distributed across multiple layers. Consider a tandem cache network of H layers, where C_H represents the origin server, and C_1 the edge server, where requests are first received. Assume, the cache size of each layer is denoted by C_j, $1 \leq j < H$. The size of the virtual "BIG" cache is denoted by $C_B := \sum_{j=1}^{H-1} C_j$. Then, any caching policy can be directly applied to this one (virtual) "BIG" cache as a single consistent strategy. Objects can be cached in any layer of the hierarchy, and moved between cache boundaries of different layers according to the caching policy (see [18] for more details). Using this "BIG" cache abstraction, the cache network can be viewed as a single "giant" (blackbox) cache with storage capability C_B, receiving multiple streams of content requests $\{\lambda_i\}$. The goal of any caching policy is to maximize the overall hit probability of the entire cache network, and minimize the load at the origin server. We now explain in details how "BIG" cache abstraction allows us to estimate the hit probability matrix for a line of caches, given the hit probability $p_i(C, \lambda_i, P)$ of object O_i under caching policy P as a function of the cache size C, and the object request rate λ_i.

Consider a line of caches with cache sizes C_h, $1 \leq h < H$. We can view the caches from layer 1 to layer j as if they represent a single virtual cache of total capacity $C_{[1:j]} := \sum_{h=1}^{j} C_h$. Thus, $p_i(C_{[1:j]}, \lambda_i, P)$ (upper) bounds the probability of serving requests for object O_i from one of the first j caches under P, and $\tilde{p}_{ij} = p_i(C_{[1:j]}, \lambda_i, P) - p_i(C_{[1:j-1]}, \lambda_i, P)$[1] yields an estimate (upper bound) of p_{ij}, the probability that requests for object O_i are served by cache C_j. Knowing the hit probability of object O_i at $L1$, we can find its probability at $L2$. Then, we repeat this process iteratively till layer L_{H-1}. Then, p_{iH} can be calculated as $1 - \sum_{j}^{H-1} p_{ij}$, which represents the percentage of requests served from the origin server. Hence, we can calculate the hit probability matrix. Thus, "BIG" cache abstraction completely avoids the aforementioned technical challenges and interdependency between cache layers, such as the filtered requests at intermediate layers. Since content objects can be stored at any layer of the cache hierarchy, the overall hit probability of each object $p_i = \sum_{h=1}^{H-1} p_{ih}$ is not affected by "BIG" cache since the percentage of requests satisfied by the cache network does not change based on the location of the object in the hierarchy, as the total caching capacity is the same. Hence, the origin server load is not affected either. However, the user's latency depends on which layer the object is served from, which depends on the caching policy. Therefore, our proposed approach can be used to estimate the hit probability matrices for different caching policies. Then, our comparison framework mentioned in Section II can be used to find the appropriate policy for the given request traffic.

Estimation Approach Validation. We evaluate our proposed

[1]This linear relationship is valid under IRM model assumption.



(a) LRU (b) q-LRU (q = 0.5)

Figure 2. Hit Probability Matrix Estimation

approach to estimate the hit probability matrix by comparing it to the simulation results for some known caching policies, and also to show how implementing these caching policies using "BIG" cache abstraction always enhances their performance when they are implemented at each cache layer independently.

LRU

$$C = \sum_i p_{hit}(i) = 1 - e^{-\lambda_i T_C} \tag{8}$$

q-LRU

$$C = \sum_i p_{hit}(i) = \frac{q(1 - e^{-\lambda_i T_C})}{e^{-\lambda_i T_C} + q(1 - e^{-\lambda_i T_C})} \tag{9}$$

From [7], and considering Poisson interarrival process for requests, we use "(8)" to calculate the hit probability of each object in a single cache of size C for LRU (always cache a copy of the requested object), and use "(9)" for q-LRU (cache a copy of the object with probability q). These equations calculate the characteristic time T_C for cache using the cache size C and the request rates for each object $\{\lambda_i\}$. Using our proposed approach, we calculate the hit probability matrix by considering the cache size of the first j layers $C_{[1:j]} := \sum_{h=1}^{j} C_h$, which would give us a new value for the characteristic time using the corresponding equation "(8)" or "(9)" according to the caching policy. Using this new characteristic time, we can calculate the probability that the object is being served from the first j layers.

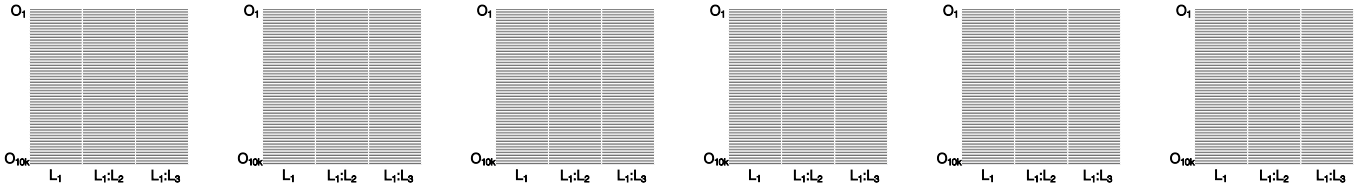
We compare the hit probability matrix calculated by our proposed approach with simulation results. We consider a line of five caches, and a collection of $10K$ unique objects. The edge cache server, and intermediate caches, have the same cache size, ranging from $[10, 50, 100, \ldots, 2500]$. User requests follow Zipf-distribution with $\alpha = 1$. We use LRU as the cache eviction policy at each cache layer, and LCE as the object replication strategy when the object is traversing a request path back to the user. The results are shown in Fig. 2a, in which *LRU(I)-sim* represents the simulation results, and *model* represents our proposed approach results. We also simulate LRU using "BIG" cache abstraction, which maintains one copy at a time at any layer due to applying LRU as a single caching strategy for the "virtual" "BIG" cache, where requested objects are always cached at the first layer $L1$ (edge server), and when $L1$ is full, the evicted objects are cached in $L2$, and so on (denoted by *LRU(B)-sim*).

The result shown in Fig. 2a indicates that *LRU(B)-sim* outperforms LRU-LCE. This is confirmed by the results in [18], which show that applying existing caching policies to a hierarchical network as a single cache, improves the performance, instead of having each cache layer taking decisions independently. In *LRU(B)-sim*, we considered the available storage as a single aggregated cache capacity which leads to caching one copy at most. Whereas other policies like *LRU(I)-sim* (LRU-LCE) results in having more than one copy of the requested object at different cache layers. Having only one copy at the cache, leaves more space for other objects to be cached, and hence improves the overall hit probability, minimizes the latency and origin server load. Our proposed estimation (*model*) consideres available storage as one single "virtual" cache with the aggregate cache capacity allowing one copy of an object in cache. In Fig. 2a it is observed that the values of (*model*) estimations and *LRU(B)-sim* matches. As *LRU(B)-sim* outperforms other policies [18], our proposed method for estimation of hit probability matrix provides an upper bound for other approximations. Similar results are shown for q-LRU in Fig. 2b.

General Cache Network. Finally, this approach can be applied to any general cache network topology using the idea discussed in Section II-C by constructing a path from each edge server to an origin server. However, to apply the "BIG" cache approach to estimate the hit matrix for each line of caches, we need to (*logically*) partition the cache resources at intermediate cache servers through which the request paths $\mathcal{L}^e$ from multiple edge servers traverse, and allot appropriate cache resources to each edge server C_e. Taking into account, the characteristics of the requests of each user populace (*i.e.*, edge server) in terms of object popularity, request interarrival distribution, ... etc, with the goal of optimizing the performance objectives. For an intermediate server C_h, $h \in \mathcal{L}^e$, let C_h^e be the portion of its cache C_h that is (logically) allotted to C_e. In other words, the cache C_h is logically partitioned into multiple pieces, C_h^e's, among the edge servers; $\sum_e C_h^e = C_h$ (here by abuse of notation we also use C_h^e and C_h to denote the cache size). Collectively, C_h^e's, $h \in \mathcal{L}^e$, form a tandem cache network with respect to the edge server C_e; its total size is $C^e := \sum_{h \in \mathcal{L}^e} C_h^e$.

For each tandem cache network corresponding to an edge server C_e, let u_i^e be the object occupancy probability, and let $U^e(\cdot)$ be a generic (*concave*) objective (or utility) function (of object occupancy or allotted cache size). We can formulate the following cache partition/allotment optimization problem. The solution[2] of this optimization problem indicates how to partition the cache resources at each intermediate cache, and allot them to the tandem cache network of each edge server to maximize the hit probability of the entire cache network. Once we have these partitions, we can apply the previously mentioned approach for each edge cache and estimate its hit probability matrix.

[2]Due to space limit, we do not elaborate on the solution of the optimization problem here.



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 3. Majorization Conditions "(3)" for the caching policies LCP & LCE in Section III

$$\underset{u_i^e \in [0,1], C_h^e}{\text{maximize}} \quad \sum_{e \in \mathcal{E}} \sum_{i}^{N} U_i^e(u_i^e)$$
$$\text{s.t.} \quad \sum_{i=1}^{N} u_i^e \leq C^e, \forall e \in \mathcal{E}; \quad \sum_{h \in \mathcal{L}^e} C_h^e = C^e, e \in \mathcal{E}; \quad (10)$$
$$\text{and} \quad \sum_{e \in \mathcal{E}} C_h^e \leq C_h, h \in H,$$

V. Evaluation

The goal of this section is to validate our proposed policy comparison framework, and show its evaluation in all the considered cases through simulation. Using common existing caching policies such as static caching, LRU, ... etc, we show that if policy P outperforms policy Q, then P *majorizes* Q according to our definition in Section II, and show that the majorization condition "(3)" is satisfied. First, we start by applying the comparison framework to a tandem line of caches. Then, we show the framework extension (discussed in Section II-C) applied to a hierarchical cache network.

A. Tandem Cache Network

We use a hierarchical network organized as a line of four caches. Edge server lies at $L1$, where user requests are first received. The origin server lies at $L4$, which serves a collection of $10K$ unique objects each of a unit size. The edge server and intermediate caches have the same cache size, which ranges from $[50, 100, \ldots, 3000]$. User requests follow Zipf-distribution with $\alpha = 0.8$. Each object has a request rate λ_i, where the aggregate request rate $\lambda = \sum_i \lambda_i = 1$. We simulated two distributions for the request intervarrival times of each object: 1) Poisson & 2) Pareto (w. parameter 2) [20]. We compare the following caching policies: static, q-LRU, LRU with different object replication strategies (LCD & LCE) [7].

Comparison Framework Validation. Using the hit probability matrices of policies P & Q, we calculate new corresponding matrices, $\hat{P}$ & $\hat{Q}$, where $\hat{p}_{kh} = \sum_{i=1}^{k} \sum_{j=1}^{h} a_i p_{ij}$, $1 \leq k \leq N, 1 \leq h \leq H$, similarly for $\hat{Q}$. $\hat{P}$ & $\hat{Q}$ represent the summation of all the possible submatrices of P & Q respectively. Finally, we define a comparison matrix X_{P-Q}, where $x_{ij} = 1$ if $\hat{p}_{ij} \geq \hat{q}_{ij}$; and 0 otherwise, where $1 \leq i \leq N, 1 \leq j \leq H$. The matrix X_{P-Q} is the representation of the majorization condition "(3)". If all elements of matrix X_{P-Q} are equal to one, then policy P *dominates* policy Q. We visualized X_{P-Q} using a heatmap, representing 1 as black, and 0 as white.

Analytically. We use the equations defined for LRU-LCE and LRU-LCP "6" & "7". The only change to the simulation settings mentioned at the beginning of this section is the number of layers, as these equations are defined for a network of 3 layers. For each policy, we calculate the characteristic time of each cache, then the hit probability matrix. We apply our comparison framework to compare their performance. The heatmap comparing LRU-LCP and LRU-LCE is shown in Fig. 3. As expected, LRU-LCP dominates LRU-LCE for the different cache sizes.

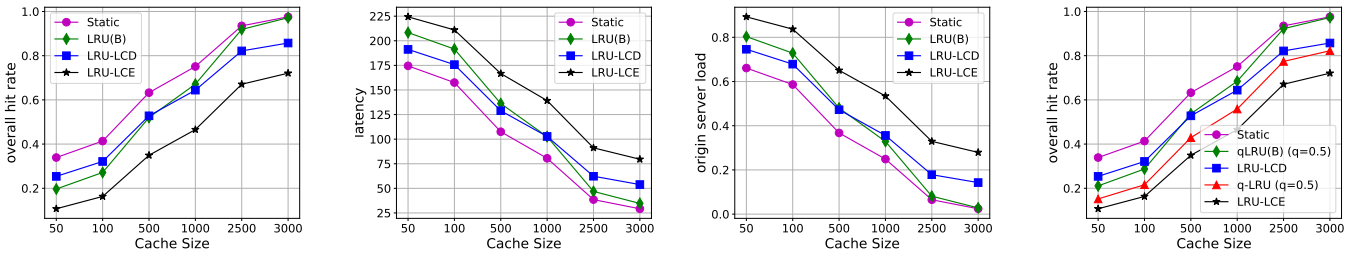
Table I
Poisson

P, Q \ CS	50	100	500	1000	2500	3000
Static, LRU(B)	✓	✓	✓	✓	✓	✓
Static, LRU-LCD	✓	✓	✓	✓	✓	✓
Static, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, LRU(B)	✓	✓	✓	✗	✗	✗
LRU-LCD, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
Static, qLRU(B)	✓	✓	✓	✓	✓	✓
Static, qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
qLRU, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, qLRU	✓	✓	✓	✓	✓	✓

Simulation. We implemented several caching policies to compare their performance using our proposed framework. Fig. 4 shows the overall performance of these caching strategies in terms of the average hit probability ($\sum_{i=1}^{N} \sum_{j=1}^{H} a_i p_{ij}$), latency and origin server load for both Poisson (top row) & Pareto (bottom row) request interarrival distributions. As expected in case of Poisson request interarrival distribution, static caching is shown to be optimal [19] if the object popularity distribution is known *a priori*, followed by LRU-LCD, LRU-LCP(q-LRU), and LRU-LCE caching policies (similar to results reported in [7]). The latency and origin server load follows the same behavior of the average hit probability, thus are not included for the other policies due to space limitations. Figs. 4a, 4d show LRU(B) & q-LRU(B), implemented using "BIG" cache abstraction [18].

LRU(B) & q-LRU(B) outperform their corresponding caching policies when implemented independently at each cache. Moreover, their performance is close to static caching performance (similar to results reported in [18]). However, as shown in [20] static caching is no longer optimal when the interarrival distribution of requests has a decreasing hazard rate (*e.g.*, when the interarrivals of requests are Pareto-distributed), we can see that LRU(B) & q-LRU(B) achieve better performance when the cache size is large as shown in Figs. 4e, 4h.

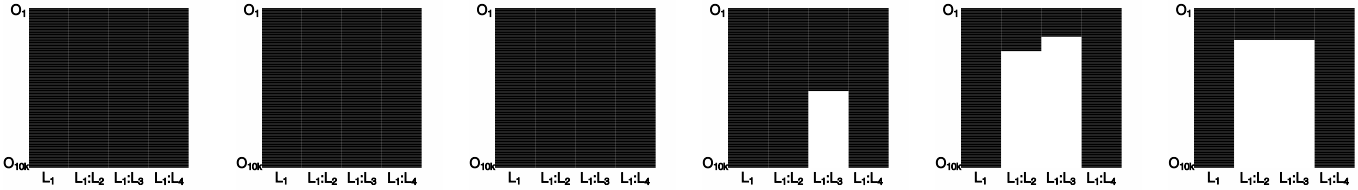
The heatmap of comparing LRU-LCD & LRU(B) in both



(a) Hit Probability (b) Latency (c) Origin Server Load (d) Hit Probability

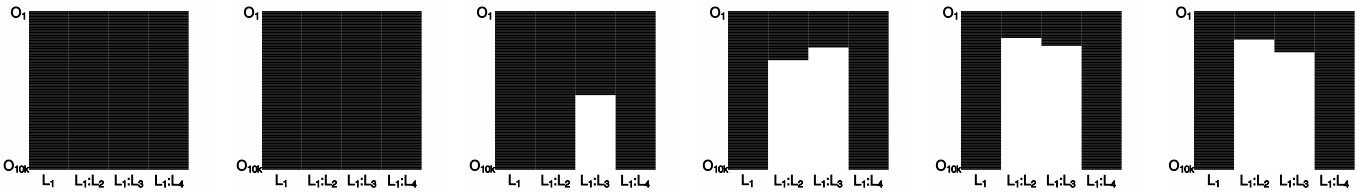
(e) Hit Probability (f) Latency (g) Origin Server Load (h) Hit Probability

Figure 4. Overall Performance for Caching Policies for a network with $N = 10K$, $H = 4$, $R = 5M$, Poisson 1st row, Pareto 2nd row



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 5. Majorization Conditions "(3)" for the caching policies LRU-LCD & LRU(B) Poisson in Fig. 4



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 6. Majorization Conditions "(3)" for the caching policies LRU-LCD & LRU(B) Pareto in Fig. 4

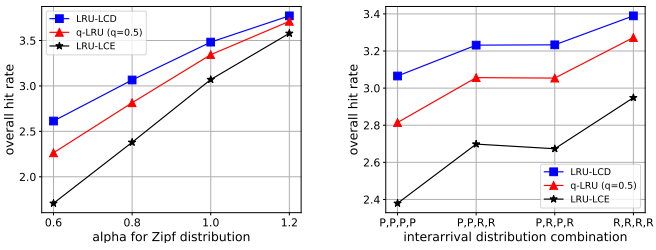
Table II
PARETO

P, Q \ CS	50	100	500	1000	2500	3000
Static, LRU(B)	✓	✓	✓	✗	✗	✗
Static, LRU-LCD	✓	✓	✓	✓	✓	✓
Static, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, LRU(B)	✓	✓	✗	✗	✗	✗
LRU-LCD, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
Static, qLRU(B)	✓	✓	✓	✗	✗	✗
Static, qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
qLRU, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, qLRU	✓	✓	✓	✓	✓	✓

cases (Poisson and Pareto distributions) are shown in Fig. 5 & Fig. 6 respectively, where they accurately reflect the dominance of LRU-LCD over LRU(B) only when the cache size is small as their corresponding average hit probability. Tables I & II summarize the comparison of every two policies P, Q for Poisson & Pareto interarrival distributions respectively for the different cache sizes, where ✓ is used if P dominates Q, and ✗ otherwise. The results in these tables reflect 100% accuracy for our proposed comparison framework in determining the dominance of the policies in comparison with respect to their simulated overall performance.

B. Tree Topology

We use a hierarchical network organized as a binary tree of three levels (*i.e.*, four edge servers). Edge servers lie at $L1$, where user requests are first received. The origin server lies at $L4$, which serves a collection of $2K$ unique objects each of a unit size. The size of the cache servers at layers [$L1$, $L2$, $L3$] are $[400, 800, 1600]$. We compare the following caching policies: q-LRU and LRU with different object replication strategies (LCD & LCE) [7].



(a) Changing Alpha (b) Changing Interarrival Distr.

Figure 7. Hierarchical Tree Cache Network

First Scenario. The distribution of the interarrival time is Poisson. We simulated 4 different values of α for zipf-distribution $[0.6, 0.8, 1.0, 1.2]$ for all the edge servers. The average hit probability is shown in Fig. 7a. We find the relationship between the three policies is still as expected, and when α increases, the average hit probability increases.

Second Scenario. We use $\alpha = 0.8$ for all edge servers, and we changed the distribution of interarrival time for the four edge servers as following: $['P, P, P, P'$, $'P, P, R, R'$, $'P, R, P, R'$, $'R, R, R, R']$, where $'P'$ refers to Poisson and $'R'$ for Pareto for the corresponding edge server. For example, $'P, P, P, P'$ means the distr. is Poisson for all edge servers in this experiment. The average hit probability is shown in Fig. 7b.

For both scenarios, we used our proposed framework to compare each two policies at each edge server. We construct a tandem cache network from this edge server to the origin server, and calculate the hit probability matrix for the objects related to this edge server. If policy P has a better performance than policy Q, policy P majorizes policy Q for each cache server in all the different cases for α.

VI. Conclusion

In this paper, we have discussed the renewed research interest in caching policies and their performance analysis as a result of the ICN architectures. Also, the additional challenges both in terms of practical cache management issues and performance analysis for cache networks. We also have discussed some of the related work done by the research community which focuses on the performance analysis of cache networks under various caching policies, and that the issue of how to evaluate and compare caching policies for a network of caches has not been adequately addressed. In this paper, we propose and develop a novel and general framework for evaluating caching policies in a hierarchical network of caches. We introduced the notion of a hit probability matrix, and employed (a generalized notion of) majorization as the basic tool for evaluating and comparing cache policies in terms of various performance metrics for a network of caches. We discussed how the framework can be applied to various existing caching policies and conduct extensive simulation-based evaluation to demonstrate the utility of our framework.

Acknowledgment: This research was supported in part by NSF grants CNS-1411636, CNS 1618339 and CNS 1617729 and a Huawei gift.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.
[2] "Named data networking," http://named-data.net/.
[3] T. Koponen and et al., "A data-oriented(and beyond)network architecture," in *ACM SIGCOMM Computer Communication Review*, 2007.
[4] S. K. Fayazbakhsh and et al., "Less pain, most of the gain: Incrementally deployable icn," in *SIGCOMM, 2013*.
[5] E. Ramadan, A. Narayanan, and Z.-L. Zhang, "Conia: Content (provider)-oriented, namespace-independent architecture for multimedia information delivery," in *ICMEW*, 2015.
[6] B. Ahlgren and et al., "A survey of information-centric networking," *IEEE Communications Magazine*, July 2012.
[7] M. Garetto, E. Leonardi, and V. Martina, "A unified approach to the performance analysis of caching systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2016.
[8] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *ACM SIGMETRICS*, 1990.
[9] P. R. Jelenkovi, "Asymptotic approximation of the move-to-front search cost distribution and least-recently used caching fault probabilities," *The Annals of Applied Probability*, 1999.
[10] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of ttl-based cache networks," in *VALUETOOLS, 2012*. IEEE.
[11] M. Dehghan, L. Massoulié, D. Towsley, D. S. Menasché, and Y. C. Tay, "A utility optimization approach to network cache design," *CoRR*, 2016.
[12] N. C. Fofack, M. Dehghan, D. Towsley, M. Badov, and D. L. Goeckel, "On the performance of general cache networks," in *ValueTools*, 2014.
[13] D. S. Berger, S. Henningsen, F. Ciucu, and J. B. Schmitt, "Maximizing cache hit ratios by variance reduction," *SIGMETRICS*, 2015.
[14] N. Gast and B. Van Houdt, "Transient and steady-state regime of a family of list-based cache replacement algorithms," *ACM SIGMETRICS Performance Evaluation Review*, 2015.
[15] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *INFOCOM, 2001*.
[16] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," in *ITC*, 2012.
[17] A. Wolman and et al., "On the scale and performance of cooperative web proxy caching," *ACM SIGOPS Operating Systems Review*, 1999.
[18] E. Ramadan, A. Narayanan, Z.-L. Zhang, R. Li, and G. Zhang, "Big cache abstraction for cache networks," in *ICDCS, 2017*. IEEE.
[19] Z. Liu, P. Nain, N. Niclausse, and D. Towsley, "Static caching of web servers," in *Multimedia Computing and Networking 1998*.
[20] A. Ferragut, I. Rodriguez, and F. Paganini, "Optimizing ttl caches under heavy-tailed demands," in *Proceedings of the 2016 ACM SIGMETRICS*.
[21] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting youtube: An active measurement study," in *INFOCOM, 2012*.
[22] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS*, 2010.
[23] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of ttl cache networks," *Performance Evaluation*, 2014.
[24] N. C. Fofack and et al., "Performance evaluation of hierarchical ttl-based cache networks," *Computer Networks*, 2014.

Neural Networks for Measurement-based Bandwidth Estimation

Sukhpreet Kaur Khangura, Markus Fidler, Bodo Rosenhahn
Department of Electrical Engineering and Computer Science, Leibniz Universität Hannover

***Abstract*—The dispersion that arises when packets traverse a network carries information that can reveal relevant network characteristics. Using a fluid-flow model of a bottleneck link with first-in first-out multiplexing, accepted probing tools measure the packet dispersion to estimate the available bandwidth, i.e., the residual capacity that is left over by other traffic. Difficulties arise, however, if the dispersion is distorted compared to the model, e.g., by non-fluid traffic, multiple bottlenecks, clustering of packets due to interrupt coalescing, and inaccurate time-stamping in general. It is recognized that modeling these effects is cumbersome if not intractable. This motivates us to explore the use of machine learning in bandwidth estimation. We train a neural network using vectors of the packet dispersion that is characteristic of the available bandwidth. Our testing results reveal that even a shallow neural network identifies the available bandwidth with high precision. We also apply the neural network under a variety of notoriously difficult conditions that have not been included in the training, such as heavy traffic burstiness, and multiple bottleneck links. Compared to two state-of-the-art model-based techniques, the neural network approach shows improved performance. Further, the neural network can effectively control the estimation procedure in an iterative implementation.**

I. Introduction

The term *available bandwidth* refers to the residual capacity of a link or a network path that is left over after the existing traffic, also referred to as *cross traffic*, is served. Knowledge of the available bandwidth benefits rate-adaptive applications and facilitates, e.g., network monitoring, detection of congested links, and load balancing. The goal of bandwidth estimation is to infer the available bandwidth of a network path using external observations of data packets, only.

Formally, given a link with capacity C and cross traffic with long-term average rate λ, where $\lambda \in [0, C]$, the available bandwidth $A \in [0, C]$ is defined as $A = C - \lambda$ [1]. The end-to-end available bandwidth of a network path is determined by its *tight link*, that is the link that has the minimal available bandwidth [2]. The tight link may differ from the *bottleneck link*, i.e., the link with the minimal capacity.

To date, a number of accepted active probing techniques and corresponding theories for available bandwidth estimation exist, e.g., [1]–[13]. These techniques use a sender that actively injects artificial *probe traffic* with a defined packet size l and inter packet gap referred to as input gap g_{in} into the network. At the receiver, the output gap of the received probe g_{out} is measured to deduce the available bandwidth.

ISBN 978-3-903176-08-9

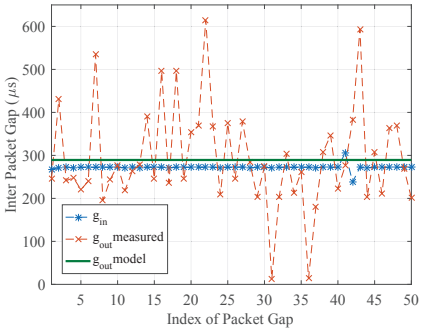


Fig. 1. Measurements of g_{in} and g_{out} compared to the fluid model. The network has a single tight link with capacity $C = 100$ Mbps and exponential cross traffic with rate $\lambda = 62.5$ Mbps. The packet size is $l = 1514$ byte.

A common assumption in bandwidth estimation is that the available bandwidth, respectively, the rate of the cross traffic does not change during a probe. Further, to simplify modeling, cross traffic is assumed to behave like fluid, i.e., effects that are due to the packet granularity of the cross traffic are neglected. Modeling a single tight link as a lossless First-In First-Out (FIFO) multiplexer of probe and cross traffic, the relation of g_{out} and g_{in} follows by an intuitive argument [1] as

$$g_{\text{out}} = \max\left\{g_{\text{in}}, \frac{g_{\text{in}}\lambda + l}{C}\right\}. \tag{1}$$

The reasoning is that during g_{in} an amount of $g_{\text{in}}\lambda$ of the fluid cross traffic is inserted between any two packets of the probe traffic, so that the probe packets may be spaced further apart. Reordering Eq. (1) gives the characteristic *gap response curve*

$$\frac{g_{\text{out}}}{g_{\text{in}}} = \begin{cases} 1 & \text{if } \frac{l}{g_{\text{in}}} \leq C - \lambda, \\ \frac{l}{g_{\text{in}}C} + \frac{\lambda}{C} & \text{if } \frac{l}{g_{\text{in}}} > C - \lambda. \end{cases} \tag{2}$$

The utility of Eq. (2) is that it shows a clear bend at $A = C - \lambda$ that enables estimating the available bandwidth using different techniques, see Sec. II. We note that the quotient of packet size and gap is frequently viewed as the data rate of the probe.

For an example, consider a tight link with capacity $C = 100$ Mbps and cross traffic with average rate $\lambda = 62.5$ Mbps. The packet size is $l = 1514$ byte, resulting in a transmission time l/C of about 120 μs. Given an input gap $g_{\text{in}} = 270$ μs, the output gap follows from Eq. (1) as $g_{\text{out}} = 290$ μs. Now, assume for the moment that C is known but λ is unknown. An active probing tool can send probes with, e.g., $g_{\text{in}} = 270$ μs to measure g_{out}. Noting that $g_{\text{out}}/g_{\text{in}} > 1$, Eq. (2) reveals the unknown $\lambda = (g_{\text{out}}C - l)/g_{\text{in}} = 62.5$ Mbps.

In practice, the observations of g_{out} are distorted for various reasons. For an example, we recorded a measurement trace of 50 pairs of g_{in} and g_{out} in the network testbed in Fig. 3 with a single tight link and the above parameters C, λ, and l, where l is the maximal size of Ethernet packets including the header. The results are shown in Fig. 1. Neglecting the cases where $g_{\mathrm{out}} < g_{\mathrm{in}}$ that are not possible in the model and ignoring large outliers, a range of samples g_{out} of about 360 μs remain that suggest concluding $\lambda \approx 90$ instead of 62.5 Mbps erroneously.

A. Challenges in Bandwidth Estimation

Relevant reasons for the distortions of g_{out} include deviations from the assumptions of the model, i.e., a lossless FIFO multiplexer with constant, fluid cross traffic as well as measurement inaccuracies, such as imprecise time-stamping:

Random cross traffic: Eq. (2) is deterministic and hence it does not define how to deal with the randomness of g_{out} that is caused by variable bit rate cross traffic. It is shown in [1] that the problem cannot be easily fixed by using the expected value $\mathsf{E}[g_{\mathrm{out}}]$ instead. In brief, this is due to the non-linearity of Eq. (2) and the fact that the location of the turning point $C - \lambda$ fluctuates if the rate of the cross traffic λ is variable. The result is a deviation that is maximal at $l/g_{\mathrm{in}} = C - \lambda$ and causes underestimation of the available bandwidth [1], [14].

Packet interference: Non-conformance with the fluid model arises due to the interaction of probes with packets of the cross traffic. In Fig. 1, two relevant examples are identified by frequent samples of g_{out} in the range of 240 and 360 μs, respectively. In contrast, the g_{out} of 290 μs, that is predicted by the fluid model, is observed rarely. To understand this effect, consider two probe packets with $g_{\mathrm{in}} = 270$ μs and note that the transmission time of a packet is 120 μs. The case $g_{\mathrm{out}} = 360$ μs occurs if two cross traffic packets are inserted between the two probe packets. Instead, if one of the two cross traffic packets is inserted in front of the probe, it delays the first probe packet, resulting in $g_{\mathrm{out}} = 240$ μs.

Packet loss: If probe packets are lost, the corresponding output gaps are void. This causes estimation bias, since packets that encounter congestion have a higher loss probability. There are few bandwidth estimation tools that consider loss, e.g., as an indication that the available bandwidth is exceeded [6].

Multiple tight links: An extensions of Eq. (1) for multiple links is derived in [3]. Yet, if cross traffic is non-fluid, the repeated packet interaction at each of the links distorts the probe gaps. Further, in case of random cross traffic, there may not be a single tight link, but the tight link may vary randomly. The consequence is an underestimation of the available bandwidth [14], [15] that is analyzed in [11], [13].

Measurement inaccuracies: Besides, there exist limitations of the accuracy due to the hardware of the hosts where measurements are taken. A possible clock offset between sender and receiver is dealt with by the use of probe gaps. A problem in high-speed networks is, however, interrupt coalescing [16], [17]. This technique avoids flooding a host with too many interrupts by grouping packets received in a short time together in a single interrupt, which distorts g_{out}.

B. State-of-the-Art Estimation Techniques

To alleviate the observed variability of the samples of g_{out}, state-of-the-art bandwidth estimation methods perform averaging of several g_{out} samples. These samples can be collected by repeated probes of two packets, so-called *packet pairs* [18], or by *packet trains* [5], [19] that consist of n consecutive packets and hence comprise $n - 1$ gaps. Further, to improve the available bandwidth estimates, statistical post-processing techniques are used, such as Kalman filtering [10], [20], majority decisions [6], averaging of the bandwidth estimates of repeated experiments [7], [8], or linear regression [4].

These techniques do, however, not overcome the basic assumptions of the deterministic fluid model in Eq. (1). While packet trains and statistical postprocessing help reduce the variability of available bandwidth estimates, they cannot resolve systematic deviations, such as the underestimation bias in case of random cross traffic and multiple tight links [1], [11], [13]. Further, it is difficult to tailor methods to specific hardware implementations that influence the measurement accuracy.

These fundamental limitations motivate us to explore the use of machine learning in available bandwidth estimation. The machine learning approach has been considered early in [21], [22] and receives increasing attention in the recent research [17], [23]. The works differ from each other with respect to their application: [21] considers the prediction of the available bandwidth from packet data traces that have been obtained in passive measurements. In contrast [17], [22], [23] use active probes to estimate the available bandwidth in NS-2 simulations [22], ultra-high speed 10 Gbps networks [17], and operational LTE networks [23], respectively.

Common to these active probing methods [17], [22], [23] is the use of packet chirps [7] that are probes of several packets sent at an increasing data rate. The rate increase is achieved either by a geometric reduction of the input gap [22], by concatenating several packet trains with increasing rates to a multi-rate probe [17], or by a linear increase of the packet size [23]. Chirps permit detecting the turning point of Eq. (2), that coincides with the available bandwidth, using a single probe. They are, however, susceptible to random fluctuations [12].

Other than chirps, [22] evaluates packet bursts that are probes of back-to-back packets and concludes that bursts are not adequate to estimate the available bandwidth. Also, [17] considers constant rate packet trains for an iterative search for the available bandwidth. Here, machine learning solves a classification problem to estimate whether the rate of a packet train exceeds the available bandwidth or not. Depending on the result, the rate of the next packet train is reduced or increased in a binary search as in [6] until the probe rate approaches the available bandwidth. The authors of [17] give, however, preference to chirp probes.

The feature vectors that are used for machine learning are generally measurements of g_{out} [22], [23] with the exception that [17] uses the Fourier transform of vectors of g_{in} and g_{out}.

Supervised learning is used and [17], [23] take advantage of today's availability of different software packages to compare the utility of state-of-the-art machine learning techniques in bandwidth estimation.

C. Contributions

In this work, we investigate how to benefit from machine learning, specifically neural networks, when using standard packet train probes for available bandwidth estimation. Compared to packet chirps, that are favored in the related works [17], [22], [23], packet trains have been reported to be more robust to random fluctuations. In fact, the implementation of a chirp as a multi-rate probe, that concatenates several packet trains with increasing rates, also benefits from this [17].

Different from multi-rate probes, packet trains are typically used in an iterative procedure that takes advantage of feedback to adapt the rate of the next packet train. Such a procedure is also proposed in [17], where machine learning is used to classify individual packet trains to control a binary search. The goal is to adapt the probe rate until it approaches the available bandwidth. In contrast, we use a feature vector that iteratively includes each additional packet train probe. This additional information enables estimating the available bandwidth directly, without the necessity that the probe rate converges to the available bandwidth. Instead of a binary search, our method chooses the probe rate next, that is expected to improve the bandwidth estimate most.

We evaluate our method in controlled experiments in a network testbed. We specifically target topologies where the assumptions of the deterministic fluid model in Eq. 1 are not satisfied, such as bursty cross traffic, and multiple tight links. For a reference, we implement two state-of-the-art model-based methods to use the same data set as our neural network-based approach.

The remainder of this paper is structured as follows. In Sec. II, we introduce the reference implementation of model-based estimation techniques. We present our neural network-based method, describe the training, and show testing results in Sec. III. In Sec. IV, we consider the estimation of available bandwidth and capacity for different tight link capacities. Our iterative neural network-based method that selects the probe rates itself is presented in Sec. V. In Sec. VI, we give brief conclusions.

II. Model-based Reference Implementations

The methods for available bandwidth estimation that are based on the fluid model of Eq. (1) essentially fall into two different categories: *iterative probing* and *direct probing*. For each of the two categories, we implement a bandwidth estimation technique that is representative of the state-of-the-art. While available bandwidth estimation tools differ significantly regarding the selection and the amount of probe traffic, our implementations are tailored to use the same database so that they provide a reference for the neural network-based method.

To reduce the variability of the measurements, a common approach is the use of constant rate packet train probes. A packet train of n packets comprises $n-1$ gaps. At the receiver, the gaps are defined as $g_{\mathrm{out}}(j) = t_{\mathrm{out}}(j+1) - t_{\mathrm{out}}(j)$ for $j = 1 \ldots n-1$, where $t_{\mathrm{out}}(j)$ is the receive time-stamp of packet j. Considering the output rate of a packet train defined as

$$r_{\mathrm{out}} = \frac{(n-1)l}{t_{\mathrm{out}}(n) - t_{\mathrm{out}}(1)} \tag{3}$$

implies averaging of the output gaps since by definition of $g_{\mathrm{out}}(j)$, Eq. (3) can be rewritten as

$$r_{\mathrm{out}} = \frac{l}{\frac{1}{n-1}\sum_{j=1}^{n-1} g_{\mathrm{out}}(j)}.$$

In case of long packet trains, stationarity, and ergodicity, the denominator converges to the mean $\overline{g_{\mathrm{out}}}$. Further, for the deterministic fluid model, $g_{\mathrm{out}}(j) = g_{\mathrm{out}}$ for $j = 1 \ldots n-1$ so that $r_{\mathrm{out}} = l/g_{\mathrm{out}}$. Similarly, the input rate for a defined g_{in} is $r_{\mathrm{in}} = l/g_{\mathrm{in}}$ and by insertion into Eq. (2) the equivalent *rate response curve* of the fluid model is obtained as

$$\frac{r_{\mathrm{in}}}{r_{\mathrm{out}}} = \begin{cases} 1 & \text{if } r_{\mathrm{in}} \leq C - \lambda, \\ \frac{r_{\mathrm{in}}+\lambda}{C} & \text{if } r_{\mathrm{in}} > C - \lambda. \end{cases} \tag{4}$$

The characteristic shape of Eq. (4) is shown in Fig. 2.

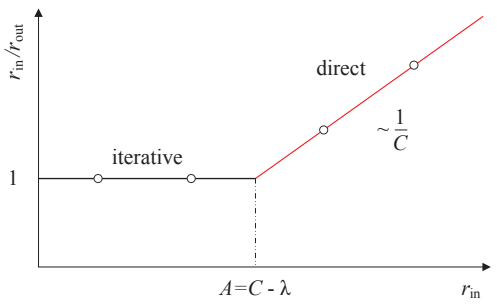


Fig. 2. Rate response curve. The turning point marks the available bandwidth.

A. Iterative probing

In brief, iterative probing techniques search for the turning point of the rate response curve by sending repeated probes at increasing rates, as long as $r_{\mathrm{in}} = r_{\mathrm{out}}$. When r_{in} reaches $C - \lambda$, the available bandwidth is saturated and increasing the probe rate r_{in} further results in self-induced congestion, so that $r_{\mathrm{in}} > r_{\mathrm{out}}$. This implies queueing at the tight link and hence increasing one way delays can be observed at the receiver.

Established iterative probing tools are, e.g., Pathload [6] and IGI/PTR [9]. Pathload adaptively varies the rates of successive packet trains r_{in} in a binary search until r_{in} converges to the available bandwidth. It uses feedback from the receiver that reports whether r_{in} exceeds the available bandwidth or not. The decision is made based on two statistical tests that detect increasing trends of the one way delay. For comparison, IGI/PTR tests whether $(r_{\mathrm{in}} - r_{\mathrm{out}})/r_{\mathrm{in}} > \Delta^{th}$, where the threshold value Δ^{th} is set to 0.1, to detect whether the probe rate exceeds the available bandwidth. Regarding the variability of the available bandwidth, Pathload reports an

available bandwidth range that is determined by the largest probe rate that did not cause self-induced congestion and the smallest rate that did cause congestion, respectively.

In our experiments, we use a dataset of equidistantly spaced r_{in} and corresponding r_{out}. We process these entries iteratively in increasing order of r_{in} and apply the threshold test of IGI/PTR [9] $(r_{\mathrm{in}} - r_{\mathrm{out}})/r_{\mathrm{in}} > \Delta^{th}$ to determine whether r_{in} exceeds the available bandwidth. We denote r_{in}^{th} the largest rate before the test detects that the available bandwidth is exceeded for the first time and report r_{in}^{th} as the available bandwidth estimate. We note that there may, however, exist $r_{\mathrm{in}} > r_{\mathrm{in}}^{th}$, where the test fails again. This may occur, for example, due to the burstiness of the cross traffic that causes fluctuations of the available bandwidth.

B. Direct probing

Instead of searching for the turning point of the rate response curve, direct probing techniques seek to estimate the parameters of the upward line segment for $r_{\mathrm{in}} > C - \lambda$. The line is determined by C and λ. If C is known, a single probe $r_{\mathrm{in}} = C$ yields a measurement of r_{out} that is sufficient to estimate $\lambda = C(C/r_{\mathrm{out}} - 1)$ from Eq. (4). Spruce [8] implements this approach. If C is also unknown, a minimum of two different probe rates $r_{\mathrm{in}} > C - \lambda$ are needed to estimate the two unknown parameters of the upward line segment of the rate response curve. This approach is taken, e.g., by TOPP [3], DietTOPP [4], and BART [10].

To implement the direct probing technique, we combine it with a threshold test to select relevant probe rates. Direct probing techniques require that $r_{\mathrm{in}} > C - \lambda$ where C and λ are unknown. We adapt a criterion from DietTOPP [4] to determine a minimum threshold $r_{\mathrm{in}}^{\min}$ that satisfies $r_{\mathrm{in}}^{\min} > C - \lambda$ and use only the probe rates $r_{\mathrm{in}} \geq r_{\mathrm{in}}^{\min}$. We use the maximal input rate in the measurement data denoted by $r_{\mathrm{in}}^{\max}$ and extract the corresponding output rate $r_{\mathrm{out}}^{\max}$. If $r_{\mathrm{in}}^{\max} > r_{\mathrm{out}}^{\max}$, it can be seen from Eq. (4) that both $r_{\mathrm{in}}^{\max} > C - \lambda$ as well as $r_{\mathrm{out}}^{\max} > C - \lambda$. Hence, we use $r_{\mathrm{in}}^{\min} = r_{\mathrm{out}}^{\max}$ as a threshold to filter out all $r_{\mathrm{in}} \leq r_{\mathrm{in}}^{\min}$. Once we have selected samples that certainly fulfill $r_{\mathrm{in}} > C - \lambda$, we use linear regression like [3], [4] to determine the upward segment of the rate response curve. The available bandwidth estimate is determined from Eq. (4) as the x-axis intercept where the regression line intersects with the horizontal line at 1, see Fig. 2.

If the assumptions of the fluid model do not hold, e.g., in case of random cross-traffic, the regression technique may occasionally fail. We filter out bandwidth estimates that can be classified as infeasible. This is the case if the slope of the regression line is so small that the intersection with 1 is on the negative r_{in} axis, implying the contradiction $A < 0$, or if the slope of the regression line is negative, implying $C < 0$.

III. Neural Network-based Method

In this section, we present our neural network-based implementation of bandwidth estimation, describe the training data sets, and show a comparison of available bandwidth estimates for a range of different network parameters.

A. Scale-invariant Implementation

We use a neural network that takes a k-dimensional vector of values $r_{\mathrm{in}}/r_{\mathrm{out}}$ as input. The corresponding r_{in} are equidistantly spaced with an increment δ_r. Hence, r_{in} is in $[\delta_r, 2\delta_r, \ldots, k\delta_r]$ that is fully defined by the parameters k and δ_r that determine the measurement resolution. Since the actual values of r_{in} do not provide additional information, they are not input to the neural network. Instead, the neural network refers to values of $r_{\mathrm{in}}/r_{\mathrm{out}}$ only by their index $i \in [1, k]$. The output of the neural network is the tuple of bottleneck capacity and available bandwidth that are also normalized with respect to δ_r, i.e., we use C/δ_r and A/δ_r, respectively. While C/δ_r and A/δ_r are not necessarily integer, they can be thought of as the index i_C and i_A where r_{in} saturates the bottleneck capacity or the available bandwidth, respectively. To obtain the actual capacity and the available bandwidth, the output of the neural network has to be multiplied by δ_r.

The normalization by δ_r achieves a neural network that is scale-invariant, since the division by δ_r replaces the units, e.g., Mbps or Gbps, by indices. Considering the fluid model in Eq. (4), the normalization of all quantities r_{in}, r_{out}, C, and λ by δ_r results in

$$\frac{r_{\mathrm{in}}}{r_{\mathrm{out}}} = \begin{cases} 1 & \text{if } i \leq i_C - i_\lambda, \\ \frac{i + i_\lambda}{i_C} & \text{if } i > i_C - i_\lambda. \end{cases} \tag{5}$$

where we used the indices $i = r_{\mathrm{in}}/\delta_r$, $i_C = C/\delta_r$, $i_\lambda = \lambda/\delta_r$, and $i_A = A/\delta_r = i_C - i_\lambda$. Eq. (5) confirms that the shape of $r_{\mathrm{in}}/r_{\mathrm{out}}$ is independent of the scale, e.g., sampling a 100 Mbps network in increments of $\delta_r = 10$ Mbps or a 1 Gbps network in increments of $\delta_r = 0.1$ Gbps reveals the same characteristic shape. The advantage of the scale-invariant representation is that the neural network requires less additional training. We note that the identity is derived under the assumptions of the fluid model and does not consider effects that are not scale-invariant such as the impact of the packet size or interrupt coalescing.

For implementation we use a $k = 20$-dimensional input vector of equidistantly sampled values of $r_{\mathrm{in}}/r_{\mathrm{out}}$. We decided for a shallow neural network consisting of one hidden layer with 40 neurons. Thus, the network comprises a 20-dimensional input vector, 40 hidden neurons and two output neurons. The output neutrons encode C/δ_r and A/δ_r.

We also explored the use of deeper networks with more hidden layers, convlayers and residual networks. However, in our setting different variants of networks did not improve the quality in our experiments. We belief, that the main reason is overfitting which is caused from the sparse amount of data used for training. When using more data or more complex settings these variants might become interesting again. Methods based on metric [24] or incremental learning [25] will also be explored in future works.

B. Training Data: Exponential Cross Traffic, Single Tight Link

We generate different data sets for training and for evaluation using a controlled network testbed. The testbed is

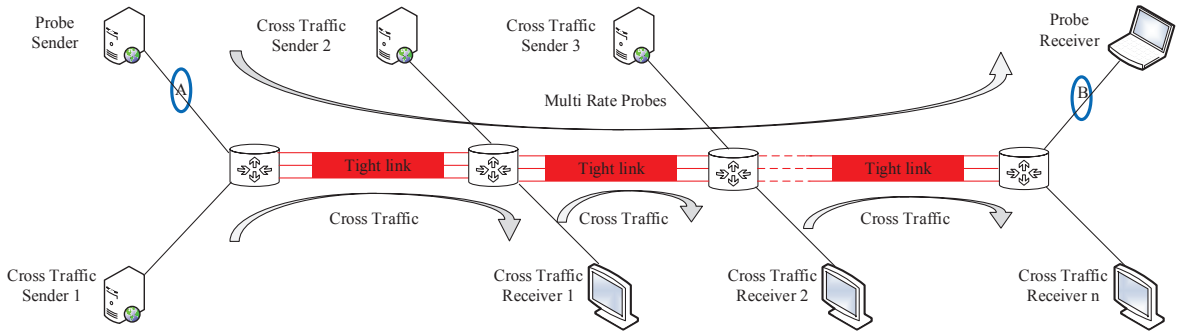


Fig. 3. Dumbbell topology set up using the Emulab and MoonGen software. A varying number of tight links with single hop-persistent cross traffic are configured. Probe-traffic is path-persistent to estimate the end-to-end available bandwidth from measurements at points A and B.

located at Leibniz Universität Hannover and comprises about 80 machines that are each connected by a minimum of 4 Ethernet links of 1 Gbps and 10 Gbps capacity via VLAN switches. The testbed is managed by the Emulab software [26] that configures the machines as hosts and routers and connects them using VLANs to implement the desired topology. We use a dumbbell topology with multiple tight links as shown in Fig. 3. To emulate the characteristics of the links, such as capacity, delay, and packet loss, additional machines are employed by Emulab. We use the MoonGen software [27] for emulation of link capacities that differ from the native Ethernet capacity. To achieve an accurate spacing of packets that matches the emulated capacity, MoonGen fills the gaps between packets by dummy frames that are discarded at the output of the link. We use the forward rate Lua script for the MoonGen API to achieve the desired forwarding rate for the transmission and reception ports of MoonGen.

Cross traffic of different types and intensities is generated using D-ITG [28]. The cross traffic is *single hop-persistent*, i.e., at each link fresh cross traffic is multiplexed. The probe traffic is *path-persistent*, i.e., it travels along the entire network path, to estimate the end-to-end available bandwidth. We use RUDE & CRUDE [29] to generate UDP probe streams. A probe stream consists of a series of k packet trains of n packets each. The k packet trains correspond to k different probe rates with a constant rate increment of δ_r between successive trains. The packet size of the probe traffic and the cross traffic is $l = 1514$ byte including the Ethernet header.

Packet timestamps at the probe sender and receiver are generated at points A and B, respectively, using libpcap at the hosts. We also use a specific endace DAG measurement card to obtain accurate reference timestamps. The timestamps are used to compute r_{in} and r_{out} for each packet train.

We generate two training data sets for a single tight link with exponential cross traffic. In data set (i) the capacity of the tight link and the access links is $C = 100$ Mbps. Exponential cross traffic with an average rate of $\lambda = 25$, 50, and 75 Mbps is used to generate different available bandwidths. In data set (ii) the capacity of the tight link is set to $C = 50$ Mbps and the exponential cross traffic has an average rate of $\lambda = 12.5$, 25, and 37.5 Mbps, respectively. In both cases the probe streams comprise packet trains of $n = 100$ packets sent at $k = 20$ different rates with rate increment $\delta_r = 5$ Mbps. For each configuration 100 repeated experiments are performed.

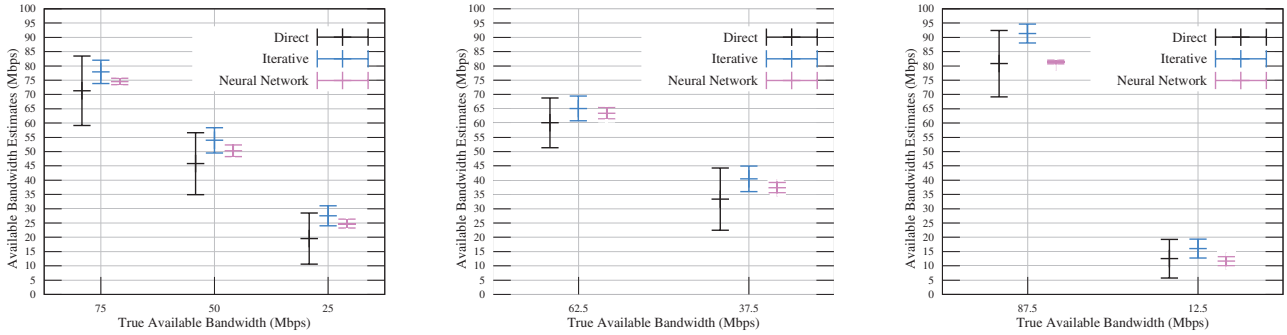
For training of the neural network, we first implement an autoencoder for each layer separately and then fine-tune the network using scaled conjugate gradient (scg). Given a regression network, we optimize the L2-error requiring approximately 1000 epochs until convergence is achieved. Training of the network (using Matlab) takes approximately 30 seconds. Due to the limited amount of training data (600 experiments overall in both training data sets), the shallow network with a small amount of hidden neurons allows training without much overfitting.

C. Evaluation: Exponential Cross Traffic, Single Tight Link

We train the neural network using the two training data sets and generate additional data sets for testing. The test data is generated for the same network configuration as the training data set (i), i.e., using exponential cross traffic of 25, 50, and 75 Mbps at a single tight link of 100 Mbps capacity. We also consider other cross traffic rates of 12.5, 37.5, 62.5, and 87.5 Mbps that have not been included in the training data set (i) to see how well the neural network interpolates and extrapolates. We repeat each experiment 100 times so that we obtain 100 bandwidth estimates for each configuration. We compare the performance of the neural network-based method with the two model-based reference implementations of an iterative and a direct estimation method. All three methods generate available bandwidth estimates from the same measurement data.

1) Testing: The testing results of the neural network-based method are summarized in Fig. 4(a) compared to the results of the direct and the iterative method. We show the average of the available bandwidth estimates with error bars that depict the standard deviation of the estimates. The variability of the available bandwidth estimates is due to a number of reasons as discussed in Sec. I-A. Particularly, the exponential cross traffic deviates from the fluid model and causes random fluctuations of the measurements of r_{out}.

The variability of the available bandwidth estimates of the direct method is comparably large and the average under-



(a) Testing (b) Interpolation (c) Extrapolation

Fig. 4. Bandwidth estimates for different cross traffic rates that have been included in the training data set (testing), that fall into the range of the training data set (interpolation), and that fall outside the range of the training data set (extrapolation). The neural network-based method provides available bandwidth estimates that exhibit little variation and have an average that matches the true available bandwidth.

estimates the true available bandwidth. The iterative method shows less variability but tends to overestimate the available bandwidth. This is a consequence of the threshold test, where a lower threshold increases the responsiveness of the test but makes it more sensitive to random fluctuations. The neural network-based method improves the bandwidth estimates significantly. The average matches the true available bandwidth and the variability is low. The good performance of the neural network is not unexpected as it has been trained for the same network parameters.

2) Interpolation: Next, we consider cross traffic of the same type, i.e., exponential, however, with a different rate that has not been included in the training data. First, we consider cross traffic rates of 37.5 and 62.5 Mbps that fall into the range of rates 25, 50, and 75 Mbps that have been used for training, hence the neural network has to interpolate. The results in Fig. 4(b) show that the available bandwidth estimates of the neural network-based method are consistent also in this case.

3) Extrapolation: Fig. 4(c) depicts available bandwidth estimates for cross traffic rates of 12.5 and 87.5 Mbps. These rates fall outside the range of rates that have been included in the training data set so that the neural network has to extrapolate. The results of the neural network-based method are nevertheless highly accurate, with a noticeable underestimation of 5 Mbps on average only in case of a true available bandwidth of 87.5 Mbps. A reason for the lower accuracy that is observed when the available bandwidth approaches the capacity is that fewer measurements are on the characteristic upward line segment, see Fig. 2 that is also used for estimation by the direct method.

D. Network Parameter Variation Beyond the Training Data

We investigate the sensitivity of the neural network with respect to a variation of network parameters that differ substantially from the training data set. Specifically, we investigate two cases that are known to be hard in bandwidth estimation. These are cross traffic with high burstiness, and networks with multiple tight links.

1) Burstiness of Cross Traffic: To evaluate how the neural network-based method performs in the presence of cross traffic with an unknown burstiness, we consider three different types of cross traffic: constant bit rate (CBR) that has no burstiness as assumed by the probe rate model, moderate burstiness due to exponential packet inter-arrival times, and heavy burstiness due to Pareto inter-arrival times with infinite variance, caused by a shape parameter of $\alpha = 1.5$. The average rate of the cross traffic is $\lambda = 50$ Mbps in all cases. As before, the tight link capacity and the access links capacities are $C = 100$ Mbps.

The burstiness of the cross traffic can cause queueing at the tight link even if the probe rate is below the average available bandwidth, i.e., if $r_{in} < C - \lambda$. This effect is not captured by the fluid model. It causes a deviation from the ideal rate response curve as depicted in Fig. 2 that is maximal at $C - \lambda$ and blurs the bend that marks the available bandwidth. The result is an increase of the variability of available bandwidth estimates as well as an underestimation bias in both direct and iterative bandwidth estimation techniques [1], [14].

Fig. 5 shows the mean and the standard deviation of 100 repeated experiments using the direct and iterative probing techniques and the neural network-based method. The average of the estimates shows a slight underestimation bias compared to the true available bandwidth if the cross traffic burstiness is increased. More pronounced is the effect of the cross traffic burstiness on the standard deviation of the bandwidth estimates. While for CBR cross traffic the estimates are close to deterministic, the variability of the estimates increases significantly if the cross traffic is bursty. The neural network, that has been trained for exponential cross traffic only, performs almost perfectly in case of CBR cross traffic and shows good results with less variability compared to the direct and iterative techniques also for the case of Pareto cross traffic.

2) Multiple Tight Links: To test the neural network with multiple tight links, we extend our network from single-hop to multi-hop as shown in Fig. 3. The path-persistent probe streams experience single hop-persistent exponential cross-traffic with average rate $\lambda = 50$ Mbps while traversing

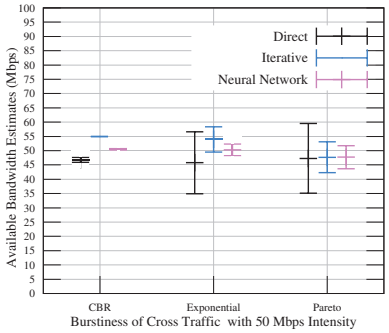


Fig. 5. Bandwidth estimates for different types of cross traffic burstiness. An increase of the burstiness causes a higher variability of the bandwidth estimates as well as an underestimation bias.

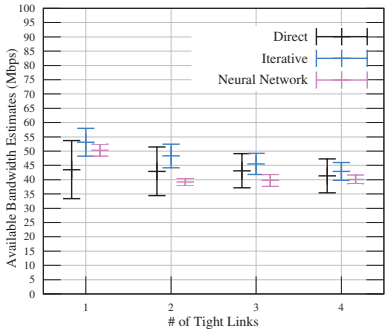


Fig. 6. Multiple tight links with capacity $C = 100$ Mbps in the presence of single hop-persistent exponential cross traffic with an average rate $\lambda = 50$ Mbps. All methods tend to underestimate the available bandwidth in case of multiple tight links.

multiple tight links of capacity $C = 100$ Mbps. The capacity of the access links is 1 Gbps.

In case of multiple tight links, the probe stream has a constant rate r_{in} with a defined input gap g_{in} only at the first link. For the following links, the input gaps have a random structure as they are the output gaps from the preceding links. At each additional link the probe stream interacts with new, bursty cross traffic. This causes lower probe output rates and results in underestimation of the available bandwidth in multi-hop networks [1], [13], [14].

In Fig. 6 we show the results from 100 repeated measurements for networks with 1 up to 4 tight links. The model-based methods, direct and iterative, as well as the neural network-based method underestimate the available bandwidth with increasing number of tight links. The reason is that the model as well as the training of the neural network consider only a single tight link. Training the neural network for multiple tight links is an interesting topic for future research. The estimates of the neural network show the least variability.

IV. Variation of the Tight Link Capacity

So far, we used test data sets that cover a tight link capacity of $C = 100$ Mbps sampled with equidistantly spaced probe rates r_{in} with an increment of $\delta_r = 5$ Mbps. Since our implementation of the neural network-based method is scale-invariant (within the limits of the fluid model), we expect that the method can perform bandwidth estimation also, e.g., in case of a tight link with $C = 50$ Mbps sampled at increments of $\delta_r = 2.5$ Mbps. If the capacity is, however, unknown, the increment δ_r cannot be adequately scaled and the measurement data will differ fundamentally. For this reason we include the training data set (ii) that is obtained for a single tight link with $C = 50$ Mbps sampled at increments of $\delta_r = 5$ Mbps. We test the neural network with data sets for $C = 50$, 100, and 200 Mbps.

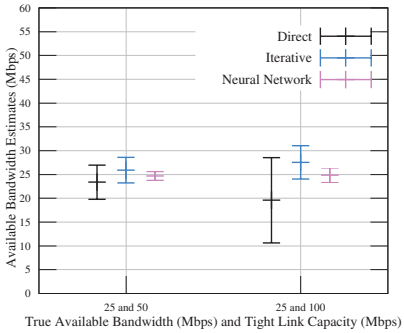


Fig. 7. Available bandwidth estimates for tight links with different capacities of $C = 50$ and 100 Mbps, respectively, and $A = 25$ Mbps available bandwidth.

A. Estimation of Available Bandwidth and Capacity

We perform testing using measurement data obtained for probe rates r_{in} with increments of $\delta_r = 5$ Mbps. The network has a single tight link with unknown available bandwidth A and unknown capacity C. In the evaluation we consider $C = 50$ and 100 Mbps and exponential cross trafic with rates $\lambda = 0.25\,C$, $0.5\,C$, and $0.75\,C$, respectively. We use the neural network to estimate both A and C.

In Fig. 7 we compare the available bandwidth estimates for $A = C - \lambda = 25$ Mbps. The results confirm that the neural network estimates the available bandwidth correctly, regardless of the capacity of the tight link. We omit further results for reasons of space and note that the neural network also estimates the capacity, i.e., 50 or 100 Mbps, with little error.

B. Capacity and Parameter Scaling

Next, we consider a proportional scaling of the network and probing parameters. In detail, the network has a single tight link with capacity $C = 50$, 100, or 200 Mbps with exponential cross traffic with rate $\lambda = 0.25\,C$, $0.5\,C$, or $0.75\,C$. The probing is performed at rate increments of $\delta_r = 2.5$, 5, and 10 Mbps, respectively. We note that only the case $C = 100$ Mbps and $\delta_r = 5$ Mbps is included in the training data set whereas the others are not. The available bandwidth estimates for $\lambda = 0.5\,C$ are presented in Fig. 8. The results confirm the utility of the scale-invariant implementation of the neural network-based method that achieves precise estimates

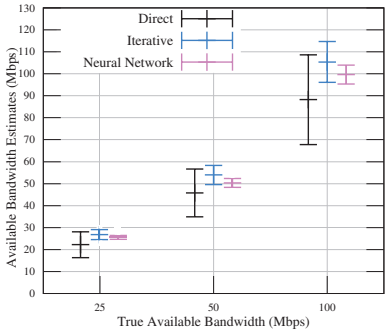


Fig. 8. Parameter scaling. Available bandwidth estimates for tight links with different capacities of $C = 50$, 100, and 200 Mbps, respectively, and $A = 0.5\,C$ available bandwidth.

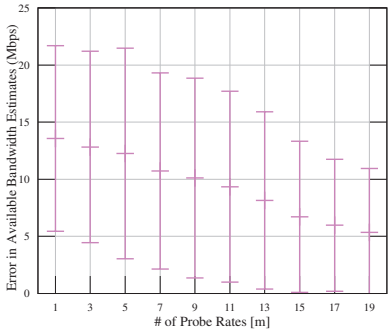


Fig. 9. Error of the available bandwidth estimates obtained for a set of m randomly selected probe rates.

in all cases. We omit showing results of the capacity estimation that was generally successful with little estimation error.

V. Iterative Neural Network-based Method

State-of-the-art iterative probing methods perform a search for the available bandwidth by varying the probe rate r_{in} until r_{in} converges to the available bandwidth. Pathload [6] uses statistical tests to determine whether r_{in} exceeds the available bandwidth or not and performs a binary search to adapt r_{in} iteratively. The recent method [17] adopts Pathload's binary search algorithm but uses machine learning instead of statistical tests to determine whether r_{in} exceeds the available bandwidth or not.

We propose an iterative neural network-based method that differs from [17] in several respects. Most importantly our method (a) determines the next probe rate by a neural network, that is trained to select the probe rate that improves the bandwidth estimate most, instead of using the binary search algorithm, and (b) it includes the information of all previous probe rates to estimate the available bandwidth instead of considering only the current probe rate. Our implementation comprises two parts. First, we train the neural network to cope with input vectors that are not fully populated. Second, we create another neural network that recommends the most beneficial probe rates.

A. Partly Populated Input Vectors

An iterative method will only use a limited set of probe rates. Correspondingly, we mark the entries of the input vector that have not been measured as invalid by setting $r_{\text{in}}/r_{\text{out}} = 0$. To obtain a neural network that can deal with such partly populated input vectors, we perform training using the training data sets (i) and (ii) where we repeatedly erase a random number of entries at random positions. When testing the neural network we erase entries in the same way.

In Fig. 9 we show the absolute error of the available bandwidth estimates that are obtained by the neural network if $m \in [1, k]$ randomly selected entries of the k-dimensional input vector are given. The bars show the average error and the standard deviation of the error. The data set used for testing is the same as the one used for Fig. 4(a) previously, i.e., $C = 100$ Mbps and $A \in [25, 50, 75]$ Mbps. We show the combined results for all values of A. The average error shows a clear improvement with increasing m whereas the standard deviation first grows slightly up to $m = 5$ before it eventually starts to improve. The reason is that for $m = 1$ the information is not sufficient to identify the two unknown parameters capacity and available bandwidth. Hence, the neural network first reports conservative estimates in the middle range. For comparison, by guessing 50 Mbps in all cases the average error is 16.6 Mbps for the given test data set. With increasing m the neural network starts to distinguish the range of $A \in [25, 50, 75]$ Mbps but tends to frequent misclassifications that can cause large errors. These misclassifications are mostly resolved when increasing m further. We observe the same trend also for the error of the capacity estimates that shows a high correlation with the error of the available bandwidth estimates. Hence, we omit showing the results.

B. Recommender Network for Probe Rate Selection

When adding entries to the partly populated input vector of the neural network, the average estimation error improves. The amount of the improvement depends, however, on the position of the a priori unknown entry that is added, as well as on the m entries that are already given, i.e., their position and value. We use a second neural network that learns this interrelation. Using this knowledge, the neural network acts as a recommender that given a partly populated input vector selects the next probe rate, i.e., the next entry, that is expected to improve the accuracy of the bandwidth estimate most. The recommender network takes the $k = 20$-dimensional input vector of values $r_{\text{in}}/r_{\text{out}}$, has 80 hidden neurons, and generates a k-dimensional output vector of estimation errors that apply if the entry $r_{\text{in}}/r_{\text{out}}$ is added at the respective position. Given the output vector, the rate r_{in} that minimizes the estimation error is selected for probing next.

Fig. 10 shows how the recommender network improves the error of the bandwidth estimates compared to the random selection of probe rates in Fig. 9. Starting at 5 selected probe rates, the average estimation error as well as the standard deviation of the error are small and adding further probe

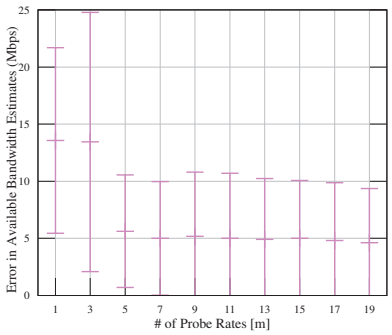


Fig. 10. Error of the available bandwidth estimates obtained for a set of m recommended probe rates.

rates improves the estimate only marginally. The reason is that certain probe rates, e.g., those on the horizontal line at $r_{\mathrm{in}}/r_{\mathrm{out}} = 1$ in Fig. 2, provide little additional information. We conclude that the recommender can effectively control the selection of probe rates to avoid those rates that contribute little. In this way, the recommender can save a considerable amount of probe traffic.

VI. Conclusion

We investigated how neural networks can be used to benefit measurement-based available bandwidth estimation. We proposed a method that is motivated by the characteristic rate response curve of a network. Our method takes a vector of ratios of equidistantly spaced probe rates at the sender and at the receiver $r_{\mathrm{in}}/r_{\mathrm{out}}$ as input to a neural network to estimate the available bandwidth and the bottleneck capacity. We use ratios of data rates and a suitable normalization to achieve an implementation that is scale-invariant with respect to the network capacity. We conducted a comprehensive measurement study in a controlled network testbed. Our results showed that neural networks can significantly improve available bandwidth estimates by reducing bias and variability. This holds true also for network configurations that have not been included in the training data set, such as different types and intensities of cross-traffic, multiple tight links, and different bottleneck capacities. To reduce the amount of probe traffic, we implemented an iterative method that varies the probe rate adaptively. The selection of probe rates is performed by a neural network that acts as a recommender. The recommender effectively selects the probe rates that reduce the estimation error most quickly.

References

[1] X. Liu, K. Ravindran, and D. Loguinov, "A queueing-theoretic foundation of available bandwidth estimation: single-hop analysis," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 918–931, 2007.

[2] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 537–549, 2003.

[3] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *IEEE Globecom*, 2000, pp. 415–420.

[4] A. Johnsson, B. Melander, and M. Björkman, "Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method," in *Swedish National Computer Networking Workshop*, 2004.

[5] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" in *IEEE INFOCOM*, 2001, pp. 905–914.

[6] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *Passive and Active Measurement Workshop*, 2002.

[7] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Passive and Active Measurement Workshop*, 2003.

[8] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *ACM Internet Measurement Conference*, 2003, pp. 39–44.

[9] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 879–894, 2003.

[10] S. Ekelin, M. Nilsson, E. Hartikainen, A. Johnsson, J.-E. Mangs, B. Melander, and M. Bjorkman, "Real-time measurement of end-to-end available bandwidth using Kalman filtering," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006, pp. 73–84.

[11] X. Liu, K. Ravindran, and D. Loguinov, "A stochastic foundation of available bandwidth estimation: Multi-hop analysis," *IEEE/ACM Transaction on Networking*, vol. 16, no. 1, pp. 130–143, 2008.

[12] J. Liebeherr, M. Fidler, and S. Valaee, "A system theoretic approach to bandwidth estimation," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1040–1053, 2010.

[13] R. Lübben, M. Fidler, and J. Liebeherr, "Stochastic bandwidth estimation in networks with random service," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 484–497, 2014.

[14] M. Jain and C. Dovrolis, "Ten fallacies and pitfalls on end-to-end available bandwidth estimation," in *ACM Internet Measurement Conference*, 2004, pp. 272–277.

[15] L. Lao, C. Dovrolis, and M. Sanadidi, "The probe gap model can underestimate the available bandwidth of multihop paths," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 29–34, 2006.

[16] R. Prasad, M. Jain, and C. Dovrolis, "Effects of interrupt coalescence on network measurements," in *Passive and Active Measurement Workshop*, 2004, pp. 247–256.

[17] Q. Yin and J. Kaur, "Can machine learning benefit bandwidth estimation at ultra-high speeds?" in *Passive and Active Measurement Conference*, 2016, pp. 397–411.

[18] S. Keshav, "A control-theoretic approach to flow control," in *Proc. ACM SIGCOMM*, Sep. 1991, pp. 3–15.

[19] V. Paxson, "End-to-end internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.

[20] Z. Bozakov and M. Bredel, "Online estimation of available bandwidth and fair share using Kalman filtering," in *IFIP Networking*, 2009.

[21] A. Eswaradass, X.-H. Sun, and M. Wu, "A neural network based predictive mechanism for available bandwidth," in *Parallel and Distributed Processing Symposium*, 2005.

[22] L.-J. Chen, "A machine learning-based approach for estimating available bandwidth," in *TENCON*, 2007, pp. 1–4.

[23] N. Sato, T. Oshiba, K. Nogami, A. Sawabe, and K. Satoda, "Experimental comparison of machine learning-based available bandwidth estimation methods over operational LTE networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2017, pp. 339–346.

[24] A. Kuznetsova, S. J. Hwang, B. Rosenhahn, and L. Sigal, "Exploiting view-specific appearance similarities across classes for zero-shot pose prediction: A metric learning approach," *Conference on Artificial Intelligence (AAAI)*, Feb. 2016.

[25] ——, "Expanding object detector's horizon: Incremental learning framework for object detection in videos," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[26] D. S. Anderson, M. Hibler, L. Stoller, T. Stack, and J. Lepreau, "Automatic online validation of network configuration in the emulab network testbed," in *IEEE International Conference on Autonomic Computing*, 2006, pp. 134–142.

[27] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *ACM Internet Measurement Conference*, Oct. 2015.

[28] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, "D-ITG distributed internet traffic generator," in *Quantitative Evaluation of Systems*, 2004, pp. 316–317.

[29] J. Laine, S. Saaristo, and R. Prior, "Real-time udp data emitter (rude) and collector for rude (crude)," 2000. [Online]. Available: https://sourceforge.net/projects/rude/

Decentralized Scheduling for Offloading of Periodic Tasks in Mobile Edge Computing

Slađana Jošilo and György Dán
ACCESS Linnaeus Center, School of Electrical Engineering and Computer Science
KTH, Royal Institute of Technology, Stockholm, Sweden E-mail: {josilo, gyuri}@kth.se

***Abstract*—Motivated by various surveillance applications, we consider wireless devices that periodically generate computationally intensive tasks. The devices aim at maximizing their performance by choosing when to perform the computations and whether or not to offload their computations to a cloud resource via one of multiple wireless access points. We propose a game theoretic model of the problem, give insight into the structure of equilibrium allocations and provide an efficient algorithm for computing pure strategy Nash equilibria. Extensive simulation results show that the performance in equilibrium is significantly better than in a system without coordination of the timing of the tasks' execution, and the proposed algorithm has an average computational complexity that is linear in the number of devices.**

I. Introduction

Mobile edge computing (MEC) is considered to become an enabler of a variety of Internet of Things (IoT) applications that are based on a pervasive deployment of wireless sensors. Examples range from water pipeline surveillance [1], through pursuit problems and discrete manufacturing [2] to body area networks [3]. Many of these applications involve the periodic collection of sensory data, which need to be processed timely to enable control decisions. Processing often requires some form of data analytics, e.g., visual analysis, which is computationally demanding.

The key advantage of MEC compared to centralized cloud infrastructures is that computational resources are located close to the network edge [4]. Thus, even though MEC infrastructures may be less resource-rich than centralized clouds, such as Microsoft Azure or AWS, due to their proximity to the sensors they may be able to provide response times that make them suitable for computation offloading for real-time applications.

The proximity of MEC resources makes low response times for individual sensors possible, but when multiple wireless sensors attempt to offload to the MEC simultaneously, the response times might increase due to contention for the communication and the computational resources [5], [6], [7]. Coordination is thus essential for maintaining low response times in the case of MEC computation offloading.

Coordination for offloading periodic tasks involves deciding whether or not to offload the computations, deciding which of the available wireless communication channels to use for offloading, and in the case of periodic tasks, it involves deciding when to collect sensory data and when to offload the computation. In addition coordination should respect that sensors may be managed by different entities, with individual interests. The resulting coordination problem not only has a huge solution space with a combinatorial structure, but it also requires consideration of the potentially diverse requirements of the sensors in terms of response time and energy consumption for performing the computation. Efficient coordination of computation offloading for wireless sensors with periodic tasks is thus a complex problem.

In this paper we address this problem by considering the allocation of cloud and wireless resources among wireless devices that generate tasks periodically. The devices can choose the time slot in which to perform their periodic task, and can decide whether to offload their computation to a cloud through one of many access points or to perform the computation locally. We provide a game theoretical treatment of the problem, and prove the existence of pure strategy Nash equilibria. Our proof provides a characterization of the structure of the equilibria, and serves as an efficient decentralized algorithm for coordinating the offloading decisions of the wireless devices. We use extensive simulations to assess the benefits of coordinated computation offloading compared to uncoordinated computation offloading where devices choose a time slot at random, and in the chosen time slot play an equilibrium allocation. Our results show that the proposed algorithm computes equilibria with good system performance in a variety of scenarios in terms of task periodicity, the number of devices and the number of access points.

The rest of the paper is organized as follows. In Section II we present the system model and the problem formulation. In Section III we present algorithmic and analytical results. In Section IV we show numerical results and in Section V we discuss related work. Section VI concludes the paper.

II. System Model and Problem Formulation

We consider a computation offloading system that consists of N devices, A acces points (APs) and a cloud service. We denote by $\mathcal{N}=\{1,2,...,N\}$ and $\mathcal{A}=\{1,2,...,A\}$ the set of devices and the set of APs, respectively. Each device generates a computationally intensive task periodically every T time units. Device i's task is characterized by the mean size D_i of the input data and by the mean number of CPU cycles L_i required to perform the computation. We make the reasonable assumption that the number X of CPU cycles required per bit can be modeled by a random variable following a Gamma distribution [8], [9], and assume $E[X]$ to be known from previous measurements. Thus, assuming independence the mean number of CPU cycles can be expressed as $L_i = D_i E[X]$.

We consider that time is partitioned into T time slots, and we denote by $\mathcal{T}=\{1,2,...,T\}$ the set of time slots. Each device can choose one time slot in which it wants

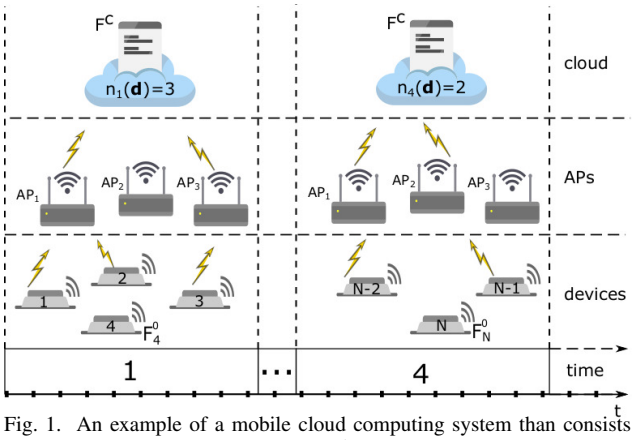


Fig. 1. An example of a mobile cloud computing system than consists of N devices, $T = 4$ time slots, and $A = 3$ APs.

to perform the computation and in the chosen time slot it can decide whether to perform the computation locally or to offload the computation to the cloud server through one of the APs. Therefore, each device $i \in \mathcal{N}$ can choose one element of the set $\mathfrak{D}_i = \{\mathcal{A} \cup \{0\}\} \times \mathcal{T}$, where 0 corresponds to local computing. We denote by $d_i \in \mathfrak{D}_i$ the decision of MU i, and refer to it as its strategy. We refer to the collection $\mathbf{d}=(d_i)_{i\in\mathcal{N}}$ as a strategy profile, and we denote by $\mathfrak{D} = \times_{i\in\mathcal{N}}\mathfrak{D}_i$ the set of all feasible strategy profiles. The considered model of homogeneous task periodicities is reasonable for surveillance of homogeneous physical phenomena, we leave the case of heterogeneous periodicities to be subject of future work.

For a strategy profile $\mathbf{d}$ we denote by $O_{(t,a)}(\mathbf{d}) = \{i | d_i = (t, a)\}$ the set of devices that offload using AP a in time slot t, and we denote by $n_{(t,a)}(\mathbf{d}) = |O_{(t,a)}(\mathbf{d})|$ the number of devices that use AP a in time slot t. Furthermore, we define the set of all devices that offload in time slot t as $O_t(\mathbf{d}) = \cup_{a\in\mathcal{A}} O_{(t,a)}(\mathbf{d})$, and the total number of devices that offload in time slot t as $n_t(\mathbf{d})=\sum_{a\in\mathcal{A}} n_{(t,a)}(\mathbf{d})$. Finally, we denote by $O(\mathbf{d}) = \cup_{t\in\mathcal{T}} O_t(\mathbf{d})$ the set of all devices that offload in strategy profile $\mathbf{d}$.

A. Local computing

In the case of local computing each device has to use its own computing resources in order to perform the computation. We consider that different devices may have different computational capabilities and we denote by F_i^0 the computational capability of device i. Furthermore, we consider that the computational capability F_i^0 of device i is independent of the chosen time slot, and hence the time that is needed for device i to perform its computation task that requires L_i CPU cycles can be expressed as

$$T_i^0 = L_i / F_i^0. \tag{1}$$

In order to express the energy consumption in the case of local computing we denote by v_i the energy consumption per CPU cycle [10], and we express the energy that device i would spend on performing a computation task that requires L_i CPU cycles as

$$E_i^0 = v_i L_i. \tag{2}$$

B. Computation offloading

In the case of computation offloading the computation is performed in the cloud, but the input data for the computation task need to be transmitted through one of the APs. In what follows we introduce our communication and computation models that describe how the wireless medium and the cloud computing resources are shared among devices that offload their tasks, respectively.

1) Communication model: We consider that the uplink rate $\omega_{i,(t,a)}(\mathbf{d})$ that device i can achieve if it offloads through AP a in time slot t is a non-increasing function $f_a(n_{(t,a)}(\mathbf{d}))$ of the number $n_{(t,a)}(\mathbf{d})$ of devices that use the same AP a in time slot t. Furthermore, we consider that each device is characterized by PHY rate $R_{i,a}$, which depends on device specific parameters such as physical layer signal characteristics and the channel conditions. Therefore, the uplink rate of device i on AP a can be different from the uplink rates of the other devices on the same AP and can be expressed as

$$\omega_{i,(t,a)}(\mathbf{d}) = R_{i,a} \times f_a(n_{(t,a)}(\mathbf{d})). \tag{3}$$

This communication model can be used to model throughput sharing mechanisms in TDMA and OFDMA based MAC protocols [11].

Given the uplink rate $\omega_{i,(t,a)}(\mathbf{d})$, the time needed for device i to transmit the input data of size D_i through AP a in time slot t can be expressed as

$$T^{tx}_{i,(t,a)}(\mathbf{d}) = D_i / \omega_{i,(t,a)}(\mathbf{d}). \tag{4}$$

We consider that every device i knows the transmit power $P_{i,a}$ that it would use to transmit the data through AP a, where $P_{i,a}$ may be determined using one of the power control algorithms proposed in [12], [13]. The transmit power $P_{i,a}$ and the transmission time $T^{tx}_{i,(t,a)}(\mathbf{d})$ determine the energy consumption of device i for transmitting the input data of size D_i through AP a in time slot t

$$E^{tx}_{i,(t,a)}(\mathbf{d}) = P_{i,a} T^{tx}_{i,(t,a)}(\mathbf{d}). \tag{5}$$

2) Computation model: We denote by F^c the computational capability of the cloud service, and we consider that the computational capability $F^c_{i,t}(\mathbf{d})$ that device i would receive from the cloud in time slot t is a non-increasing function $f_i(n_t(\mathbf{d}))$ of the total number $n_t(\mathbf{d})$ of devices that offload in time slot t

$$F^c_{i,t}(\mathbf{d}) = F^c \times f_i(n_t(\mathbf{d})). \tag{6}$$

Therefore, the time needed for performing device i's task in the cloud may be different in different time slots, and given the number L_i of CPU cycles needed for the computation task it can be expressed as

$$T^{exe}_{i,t}(\mathbf{d}) = L_i / F^c_{i,t}(\mathbf{d}). \tag{7}$$

We consider that a single time slot is long enough for performing each user's task both in the case of local computing and in the case of computation offloading. This assumption is reasonable in the case of real time applications, where the worst-case task completion time must be less than a fraction of the periodicity.

Figure 1 shows an example of a mobile cloud computing system where devices can choose one slot out of four time slots to perform the computation. In the case of computation offloading, each device in the chosen time slot can offload its task to the cloud through one of three APs, e.g., in time slot 1 devices 1 and 2 offload their tasks through AP 1, device 3 offloads its task through AP 3, and device 4 performs the computation locally.

C. Cost Model

We consider that devices are interested in minimizing a linear combination of their computing time and their energy consumption, and denote by $0 \leq \gamma_i^T, \gamma_i^E \leq 1$ the corresponding weights, respectively. We can then express the cost of device i in the case of local computation as

$$C_i^0 = \gamma_i^T T_i^0 + \gamma_i^E E_i^0. \quad (8)$$

Similarly, we can express the cost of device i in the case of offloading through AP a in time slot t as

$$C_{i,(t,a)}^c(\mathbf{d}) = \gamma_i^T (T_{i,t}^{exe}(\mathbf{d}) + T_{i,(t,a)}^{tx}(\mathbf{d})) + \gamma_i^E E_{i,(t,a)}^{tx}(\mathbf{d}). \quad (9)$$

In (9) we made the common assumption that the time needed to transmit the result of the computation from the cloud service to the device can be neglected [5], [14], [15], [7], because for many applications (e.g., object recognition, tracking) the size of the output data is significantly smaller than the size D_i of the input data. We can thus express the cost of device i in strategy profile $\mathbf{d}$ as

$$C_i(\mathbf{d}) = \sum_{d_i \in \mathcal{T} \times \{0\}} \mathbf{1}_{(t,0)}(d_i) \cdot C_i^0 + \sum_{d_i \in \mathcal{T} \times \mathcal{A}} \mathbf{1}_{(t,a)}(d_i) \cdot C_{i,(t,a)}^c(\mathbf{d}), \quad (10)$$

where $\mathbf{1}_{(t,d)}(d_i)$ is the indicator function, i.e., $\mathbf{1}_{(t,d)}(d_i) = 1$ if $d_i = (t,d)$ and $\mathbf{1}_{(t,d)}(d_i) = 0$ otherwise.

D. Multi-slot computation offloading game

We consider that the objective of each device is to minimize its own total cost (10), i.e., to find a strategy

$$d_i^* \in \arg\min_{d_i \in \mathfrak{D}_i} C_i(d_i, d_{-i}), \quad (11)$$

where $C_i(d_i, d_{-i})$ is the cost of device i if it chooses strategy d_i given the strategies d_{-i} of the other devices. Since devices may be autonomous entities with individual interests, we model the problem as a strategic game $\Gamma = < \mathcal{N}, (\mathfrak{D}_i)_i, (C_i)_i >$, in which the set of players is the set of devices (we use these two terms interchangeably). We refer to the game as the *multi-slot computation offloading game* (MSCOG). The MSCOG is a player specific network congestion game, as illustrated in Fig. 2.

Our objective is to answer the fundamental question whether there is a strategy profile from which no device would want to deviate, i.e., a pure strategy Nash equilibrium.

Definition 1. A pure strategy Nash equilibrium (NE) is a strategy profile $\mathbf{d}^*$ in which all players play their best replies to each others' strategies, that is,

$$C_i(d_i^*, d_{-i}^*) \leq C_i(d_i, d_{-i}^*), \forall d_i \in \mathfrak{D}_i, \forall i \in \mathcal{N}.$$

Given a strategy profile $d = (d_i', d_{-i})$, an *improvement step* of device i is a strategy d_i' such that $C_i(d_i', d_{-i}) < C_i(d_i, d_{-i})$. A *best improvement* step is an improvement step that is a best reply. A *(best) improvement path* is a sequence of strategy profiles in which one device at a time changes its strategy through performing a *(best) improvement step*. We refer to the device that makes the best improvement step as the *deviator*. Observe that no device can perform a best improvement step in a NE.

III. Computing Equilibria

A. Single time slot ($T = 1$)

We start with considering the case $T = 1$, i.e., a single time slot.

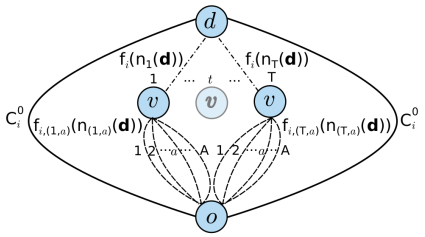


Fig. 2. Network model of the MSCOG.

Theorem 1. *The MSCOG for $T = 1$ possesses a pure strategy Nash equilibrium.*

Proof. We prove the result by showing that the game is best response equivalent to a player specific congestion game $\tilde{\Gamma}$ on a parallel network, i.e., a singleton player specific congestion game [16]. Observe that if for $T = 1$ we contract the edge (v, d) in the network shown in Fig. 2, i.e., if we replace the edge (v, d) and its two end vertices v and d by a single vertex, then we obtain a parallel network. Let us define the local computation cost of player i in $\tilde{\Gamma}$ as $\tilde{C^0}_i(N - n_1(\mathbf{d})) = C_i^0 - f_i(1 + n_1(\mathbf{d})) + c$, and the cost of offloading through AP a as $\tilde{f}_{i,a}(n_{(1,a)}(\mathbf{d})) = f_{i,a}(n_{(1,a)}(\mathbf{d})) + c$, where c is a suitably chosen constant to make all costs non-negative. Observe that due to the contraction of the edge (v, d) the offloading cost is $\tilde{C^c}_{i,a} = C_{i,a}^c - f_i(n_1(\mathbf{d}))$, and thus the difference between the cost function of player i in $\tilde{\Gamma}$ and that in Γ only depends on the strategies of the other players. This in fact implies that $\tilde{\Gamma}$ and Γ are best-response equivalent, and thus they have identical sets of pure strategy Nash equilibria. Since $\tilde{\Gamma}$ is a singleton player specific congestion game, it has a NE, and so does Γ, which proves the result. □

Furthermore, a Nash equilibrium of the MSCOG can be found in polynomial time.

Corollary 1. *Consider a MSCOG with $T = 1$ and N players. Let $\mathbf{d}^*$ be a Nash equilibrium of the game, and consider that a new player is added to the game. Then there is a sequence of best responses that leads to a NE.*

Proof. The result follows from the best response equivalence to $\tilde{\Gamma}$, and from the proof of Theorem 2 in [17]. □

Unfortunately, the contraction technique used in the proof of Theorem 1 cannot be applied for $T > 1$, as the resulting game would no longer be a congestion game.

B. Multiple time slots ($T \geq 1$)

In order to answer the question for $T \geq 1$ we first show that if a pure strategy NE exists for $T \geq 1$ then its structure cannot be arbitrary.

Theorem 2. *Assume that $\mathbf{d}^*$ is a NE of the MSCOG with $T \geq 1$. Then the following must hold*
(i) $\min_{t' \in \mathcal{T}} n_{t'}(\mathbf{d}^*) \leq n_t(\mathbf{d}^*) \leq \min_{t' \in \mathcal{T}} n_{t'}(\mathbf{d}^*) + 1$ *for* $\forall t, t' \in \mathcal{T}$,
(ii) if $n_t(\mathbf{d}^*) = n_{t'}(\mathbf{d}^*) + 1$ *for some* $t' \in \mathcal{T} \setminus \{t\}$, *then* $n_{(t,a)}(\mathbf{d}^*) \leq n_{(t',a)}(\mathbf{d}^*) + 1$ *for every AP* $a \in \mathcal{A}$, *and*
(iii) if $n_{(t,a)}(\mathbf{d}^*) = n_{(t',a)}(\mathbf{d}^*) - k$ *for* $k > 1$ *and* $t' \neq t$, *then* $n_{t'}(\mathbf{d}^*) \leq n_t(\mathbf{d}^*) \leq n_{t'}(\mathbf{d}^*) + 1$.

Proof. Clearly, all statements hold for $T = 1$. Assume that $T > 1$ and $\exists t, t' \in \mathcal{T}$ such that $n_t(\mathbf{d}^*) > n_{t'}(\mathbf{d}^*) + 1$. Then $\exists a \in \mathcal{A}$ such that $n_{(t,a)}(\mathbf{d}^*) \geq n_{(t',a)}(\mathbf{d}^*) + 1$. Therefore, player $i \in O_{(t,a)}(\mathbf{d}^*)$ could decrease her cost by changing

$\mathbf{d}^* = MB(\mathcal{N}, \mathcal{TA}, F^c, F_i^0)$

Let $N \leftarrow 1$
for $N = 1 \ldots |\mathcal{N}|$ **do**
  Let $A' \leftarrow \emptyset$ /*APs with decreased number of offloaders*/
  Let $i \leftarrow N$
  $d_i^* = \arg\min_{d \in \mathfrak{D}_i} C_i(d, \mathbf{d}^*(N-1))$
  Let $\mathbf{d} \leftarrow (d_i^*, \mathbf{d}^*(N-1))$
  if $d_i^* = (t, a)$ s.t. $a \in \mathcal{A}$ **then**
    /*Players $j \in O_{(t,a)}(\mathbf{d})$ play best replies*/
    $(\mathbf{d}', t', A') = DPD(\mathbf{d}, \mathbf{d}^*(N-1), (t, a), A')$
    if $\exists j \in O_t(\mathbf{d}'), \exists d_j \in \mathfrak{D}_j$ s.t. $C_j(d_j, d'_{-j}) < C_j(d'_j, d'_{-j})$ **then**
      /*Players $j \in O_t(\mathbf{d}')$ play best replies*/
      $d_j = \arg\min_{d \in \mathfrak{D}_j} C_j(d, d'_{-j})$
      Let $\mathbf{d} \leftarrow (d_j, d'_{-j})$, Update A'
      if $\exists i \in O_{d_i}(\mathbf{d}), d_i \neq \arg\min_{d \in \mathfrak{D}_i} C_i(d, d_{-i}) \notin A'$ **then**
        Let $(t, a) \leftarrow d_j$, **go to** 9
      else
        Let $\mathbf{d}' \leftarrow \mathbf{d}$
      end if
    end if
    if $A' \neq \emptyset$ **then**
      /*Players $j \in O(\mathbf{d}') \cup L(\mathbf{d}')$ play best replies*/
      $(\mathbf{d}, (t, a), A') = SID(\mathbf{d}', A')$
      if $\exists i \in O_{(t,a)}(\mathbf{d}), d_i \neq \arg\min_{d \in \mathfrak{D}_i} C_i(d, d_{-i}) \notin A'$ **then**
        go to 9
      else if $\exists i \in O(\mathbf{d}) \cup L(\mathbf{d}), d_i \neq \arg\min_{d \in \mathfrak{D}_i} C_i(d, d_{-i}) \in A'$ **then**
        Let $\mathbf{d}' \leftarrow \mathbf{d}$, **go to** 22
      end if
    end if
  end if
  Let $\mathbf{d}^*(N) \leftarrow \mathbf{d}'$
end for
return $\mathbf{d}^*(N)$

Fig. 3. Pseudo code of the MB algorithm.

the strategy to offloading through AP a in time slot t'. This contradicts $\mathbf{d}^*$ being a NE and proves (i).

We continue by proving (ii). Assume that there is an AP a such that $n_{(t,a)}(\mathbf{d}^*) > n_{(t',a)}(\mathbf{d}^*) + 1$ holds. Since $n_t(\mathbf{d}^*) = n_{t'}(\mathbf{d}^*) + 1$, we have that player $i \in O_{(t,a)}(\mathbf{d}^*)$ could decrease her cost by changing the strategy from (t,a) to (t',a). This contradicts $\mathbf{d}^*$ being a NE and proves (ii).

Finally, we prove (iii). First, assume that $n_t(\mathbf{d}^*) < n_{t'}(\mathbf{d}^*)$. Since $n_{(t,a)}(\mathbf{d}^*) < n_{(t',a)}(\mathbf{d}^*) - 1$, we have that player $i \in O_{(t',a)}(\mathbf{d}^*)$ could decrease her cost by changing the strategy from (t',a) to (t,a). This contradicts $\mathbf{d}^*$ being a NE and proves that $n_t(\mathbf{d}^*) \geq n_{t'}(\mathbf{d}^*)$. Second, assume that $n_t(\mathbf{d}^*) > n_{t'}(\mathbf{d}^*) + 1$ holds. Since $n_{(t,a)}(\mathbf{d}^*) < n_{(t',a)}(\mathbf{d}^*) - 1$, there is at least one AP $b \neq a$ such that $n_{(t,b)}(\mathbf{d}^*) \geq n_{(t',b)}(\mathbf{d}^*) + 1$, and thus player $i \in O_{(t,b)}(\mathbf{d}^*)$ could decrease her cost by changing the strategy to (t', b). This contradicts $\mathbf{d}^*$ being a NE and proves that $n_t(\mathbf{d}^*) \leq n_{t'}(\mathbf{d}^*) + 1$ must hold. □

In what follows we prove our main result concerning the existence of an equilibria in general case.

Theorem 3. *The MSCOG for $T \geq 1$ possesses a pure strategy Nash equilibrium.*

We provide the proof in the rest of the section.

C. The MyopicBest (MB) Algorithm

We prove Theorem 3 using the MB algorithm, shown in Fig. 3. The MB algorithm adds players one at a time, and lets them play their best replies given the other players' strategies. Our proof is thus based on an induction in the number N of players, and starts with the following result.

$(\mathbf{d}, t, A') = DPD(\mathbf{d}, \mathbf{d}^*(N-1), (t, a), A')$

/*Players that want to stop to offload*/
$D'_1 = \{j | d_j = (t, a), (t, 0) = \arg\min_{d \in \mathfrak{D}_j} C_j(d, d_{-j})\}$
/*Player that want to change offloading strategy*/
$D'_2 = \{j | d_j = (t, a), (t', b) = \arg\min_{d \in \mathfrak{D}_j} C_j(d, d_{-j}) \notin A', (t, a) \neq (t', b)\}$
while $|D'_1 \cup D'_2| > 0$ **do**
  /*Players that want to stop to offload have priority*/
  if $|D'_1| > 0$ **then**
    Take $i \in D'_1$
    $d_i = (t, 0)$
  else
    Take $i \in D'_2$
    Let $d_i = \arg\min_{d \in \mathcal{T} \times \mathcal{A}} C_i(d, d_{-i})$
    Let $(t, a) \leftarrow d_i$
  end if
  Let $\mathbf{d} \leftarrow (d_i, d_{-i})$
  Update A', D'_1, D'_2
end while
return $(\mathbf{d}, t, A')$

Fig. 4. Pseudo code of the DPD algorithm.

Theorem 4. *The MB algorithm terminates in a NE for $N \leq T$.*

Proof. It is easy to see that if a strategy profile $\mathbf{d}^*(N)$ is a NE for $N \leq T$ then by Theorem 2 there is at most one player per time slot, and the MB algorithm computes such a strategy profile. □

We continue by considering the case $N > T$. Let us assume that for $N - 1 \geq T$ there is a NE $\mathbf{d}^*(N-1)$ and that upon induction step N a new player i enters the game and plays her best reply d_i^* with respect to $\mathbf{d}^*(N-1)$. After that, players can make best improvement steps one at a time starting from the strategy profile $\mathbf{d} = (d_i^*, \mathbf{d}^*(N-1))$. If $d_i^* = (t, 0)$, then $n_{(t,a)}(\mathbf{d}) = n_{(t,a)}(\mathbf{d}^*(N-1))$ holds for every $(t, a) \in \mathcal{T} \times \mathcal{A}$, and thus $\mathbf{d}$ is a NE. Otherwise, if $d_i^* = (t, a)$, for some $a \in \mathcal{A}$, some players $j \in O_{(t,a)}(\mathbf{d})$ may have an incentive to make an improvement step because their communication and cloud computing costs have increased, and some players $j \in O_t(\mathbf{d}) \setminus O_{(t,a)}(\mathbf{d})$ may have an incentive to make an improvement step because their cloud computing cost has increased. Among these players, the MB algorithm allows players $j \in O_{(t,a)}(\mathbf{d})$ to perform best improvement steps, using the *DoublePokeDeviator* (DPD) algorithm shown in Fig. 4. There are two types of players that can make a best improvement step using the DPD algorithm. The first type are players $j \in O_{(t,a)}(\mathbf{d})$ for which a best reply is to stop to offload. The second type are players $j \in O_{(t,a)}(\mathbf{d})$ for which a best reply is an offloading strategy $(t', b) \in \mathcal{T} \times \mathcal{A} \setminus \{(t, a)\}$ for which the number of offloaders in $\mathbf{d}$ is not smaller than the number of offloaders in the NE $\mathbf{d}^*(N-1)$. The DPD algorithm allows either one player of the first type, or one player of the second type to perform a best improvement step, and as we show next it terminates in a finite number of steps.

Proposition 1. *Let $\mathbf{d}$ be a strategy profile in which there is at least one player $j \in O_{(t,a)}(\mathbf{d})$ that can be chosen by the DPD algorithm. Then the length of a best improvement path generated by the DPD algorithm is at most $N - 1$.*

Proof. Let us denote by $\mathbf{d}'$ a strategy profile after a player $j \in O_{(t,a)}(\mathbf{d})$ performs its best improvement step. First, observe that if player j's best improvement step is to stop to offload, then the DPD algorithm terminates since it

allows only players that play the same strategy as the last deviator to perform best improvement steps. Furthermore, if $\mathbf{d} = (d_i^*, \mathbf{d}^*(N-1))$, then $n_{(t,a)}(\mathbf{d}') = n_{(t,a)}(\mathbf{d}^*(N-1))$ for every $(t,a) \in \mathcal{T} \times \mathcal{A}$, and thus $\mathbf{d}'$ is a NE.

Otherwise, if player j's best improvement step is $(t',b) \in \mathcal{T} \times \mathcal{A} \setminus \{(t,a)\}$, then $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d}) + 1$ holds, and we can have one of the following: (1) there is no player $j' \in O_{(t',b)}(\mathbf{d})$ that wants to deviate from (t',b), (2) there is a player $j' \in O_{(t',b)}(\mathbf{d})$ that wants to deviate from (t',b).

If case (1) happens then the DPD algorithm terminates, because there is no player that plays the same strategy as the last deviator and that can decrease its cost using the DPD algorithm. Otherwise, if case (2) happens then a new best improvement step can be triggered, which will bring the system to a state where $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d})$ holds.

In what follows we show that none of the players that has changed its offloading strategy in one of the previous best improvement steps would have an incentive to deviate again. Let us consider a player j' that changed its strategy from (t',b) to another offloading strategy, and let us assume that in one of the subsequent best improvement steps one of the players changes its offloading strategy to (t',b), and thus it brings the system to a state where $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d}) + 1$ holds. We observe that player j that has changed its strategy from (t,a) to (t',b) before player j' deviated from (t',b) would have no incentive to deviate from its strategy (t',b) after a new player starts offloading through AP b in time slot t'. This is because (t',b) was its best response while player j' was still offloading through AP b in time slot t', i.e, while $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d}) + 1$ was true. Therefore, a new best improvement step can be triggered only if there is another player that wants to change from (t',b) to another offloading strategy. If this happens, $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d})$ will hold again, and thus the maximum number of players that offload through AP b in time slot t' will be at most $n_{(t',b)}(\mathbf{d})+1$ in all subsequent best improvement steps. Consequently, player j would have no incentive to leave AP b in time slot t' in the subsequent steps. Therefore, each player deviates at most once in a best improvement path generated by the DPD algorithm, and thus the algorithm terminates in at most $N-1$ best improvement steps, which proves the proposition. □

The DPD algorithm may be called multiple times during the execution of the MB algorithm, but as we show next for any fixed N, it is called a finite number of times.

Proposition 2. *The DPD algorithm is executed a finite number of times for any particular N.*

Proof. Let us assume that the DPD algorithm has been called at least once during the execution of the MB algorithm, and let us denote by $\mathbf{d}'$ the most recent strategy profile computed by the DPD algorithm. Now, let us assume that in the next best improvement step generated by the MB algorithm a player $i \in O(\mathbf{d}') \cup L(\mathbf{d}')$ changes its strategy to $(t,a) \in \mathcal{T} \times \mathcal{A}$. Starting from a strategy profile $\mathbf{d} = ((t,a), d'_{-i})$ players $j \in O_{(t,a)}(\mathbf{d})$ are allowed to perform the next best improvement step using the DPD algorithm.

Observe that players $j' \in O_{(t,a)}(\mathbf{d}')$ that in the previous best improvement steps changed their strategy to (t,a) using the DPD algorithm and triggered one of the players to leave the same strategy (t,a) would have no incentive to perform a best improvement step using the DPD algorithm. This is because the previous deviators $j' \in O_{(t,a)}(\mathbf{d}')$ brought $n_{(t,a)}(\mathbf{d}')$ to its maximum, that is to $n_{(t,a)}(\mathbf{d}^*(N-1)) + 1$, which decreased again to $n_{(t,a)}(\mathbf{d}^*(N-1))$ after the next deviator left strategy (t,a). Since the number of previous deviators $j' \in O_{(t,a)}(\mathbf{d}')$ that have no incentive to perform a new best improvement step using the DPD algorithm increases with every new best improvement path generated by the DPD algorithm, players will stop performing best improvement steps using the DPD algorithm eventually, which proves the proposition. □

So far we have proven that the DPD algorithm generates a finite number of finite best improvement paths. In the following we use this result for proving the convergence of the MB algorithm.

Proof of Theorem 3. We continue with considering all conditions under which the DPD algorithm may have terminated. First, let us assume that the last deviator's best improvement step is a strategy within time slot t'. The proof of Proposition 1 shows that the DPD algorithm terminates if one of the following happens: (i) starting from a strategy profile $\mathbf{d} = (d_i^*, \mathbf{d}^*(N-1))$ all players performed their best improvement steps, (ii) some players did not deviate and the last deviator's strategy was $(t',0)$, i.e., the last deviator changed to local computing in time slot t', (iii) some players did not deviate and there was no player that wanted to change from the last deviator's strategy $(t',b) \in \mathcal{T} \times \mathcal{A}$.

Let us first consider case (i), and the last deviator that performed its best improvement step. If its best improvement step was to stop to offload, $n_{(t,a)}(\mathbf{d}') = n_{(t,a)}(\mathbf{d}^*(N-1))$ holds for every $(t,a) \in \mathcal{T} \times \mathcal{A}$. Otherwise, if a best improvement step of the last deviator was to change its offloading strategy to (t',b), we have that $n_{(t,a)}(\mathbf{d}') \geq n_{(t,a)}(\mathbf{d}^*(N-1))$ for every $(t,a) \in \mathcal{T} \times \mathcal{A}$, where the strict inequality holds only for (t',b), and $n_{(t',b)}(\mathbf{d}') = n_{(t',b)}(\mathbf{d}^*(N-1)) + 1$. Since there is no offloading strategy for which the number of offloaders is less than the number of offloaders in the NE $\mathbf{d}^*(N-1)$, there is no player $j \in O(\mathbf{d}')$ that can decrease its offloading cost. Furthermore, there is no player that wants to change its strategy from local computing to offloading, and thus a strategy profile computed by the DPD algorithm is a NE.

If case (ii) or case (iii) happen the MB algorithm allows players that offload in the same time slot as the last deviator to perform any type of best improvement steps. Furthermore, if case (ii) happens and there are no APs with decreased number of offloaders compared with the NE $\mathbf{d}^*(N-1)$, i.e., $n_{(t,a)}(\mathbf{d}') = n_{(t,a)}(\mathbf{d}^*(N-1))$ holds for every $(t,a) \in \mathcal{T} \times \mathcal{A}$, then the strategy profile $\mathbf{d}'$ computed by the DPD algorithm is a NE. Observe that $n_{(t,a)}(\mathbf{d}') = n_{(t,a)}(\mathbf{d}^*(N-1))$ holds for every $(t,a) \in \mathcal{T} \times \mathcal{A}$ if strategy profile $\mathbf{d}'$ is obtained by the DPD algorithm starting from strategy profile $\mathbf{d} = (d_i^*, \mathbf{d}^*(N-1))$.

Otherwise, if case (ii) happens such that there is a strategy $(t,a) \in \mathcal{T} \times \mathcal{A}$ for which $n_{(t,a)}(\mathbf{d}') < n_{(t,a)}(\mathbf{d}^*(N-1))$ holds, then players $j \in O_{t'}(\mathbf{d}')$ that offload in the same time slot as the last deviator may want to change their offloading strategy to (t,a). Let us assume that there is a player

$j \in O_{t'}(\mathbf{d}')$ that wants to change its offloading strategy to (t,a) and let us denote by $\mathbf{d}$ a resulting strategy profile. Since $n_{(t,a)}(\mathbf{d}) = n_{(t,a)}(\mathbf{d}') + 1$ and $n_t(\mathbf{d}) = n_t(\mathbf{d}') + 1$ hold, some players $j \in O_{(t,a)}(\mathbf{d})$ may want to perform a best improvement step using the DPD algorithm, which can happen only a finite number of times accoring to Proposition 2.

We continue the analysis by considering case (iii). Observe that if there is a strategy (t,a) for which $n_{(t,a)}(\mathbf{d}') < n_{(t,a)}(\mathbf{d}^*(N-1))$ players $j \in O_{t'}(\mathbf{d}')$ that offload in the same time slot as the last deviator may want to change their offloading strategy to (t,a). Furthermore, players $j \in O_{t'}(\mathbf{d}') \setminus O_{(t',b)}(\mathbf{d}')$ may want to stop to offload or to change to any offloading strategy $(t,a) \in \mathcal{T} \times \mathcal{A} \setminus \{(t',b)\}$ since their cloud computing cost increased. Let us assume that there is a player $j \in O_{t'}(\mathbf{d}')$ that wants to change its offloading strategy to $(t,a) \in \mathcal{T} \times \mathcal{A} \setminus \{(t',b)\}$ and let us denote by $\mathbf{d}$ the resulting strategy profile. Since $n_{(t,a)}(\mathbf{d}) = n_{(t,a)}(\mathbf{d}') + 1$ and $n_t(\mathbf{d}) = n_t(\mathbf{d}') + 1$ hold, some players $j \in O_{(t,a)}(\mathbf{d})$ may want to perform a best improvement step using the DPD algorithm, which can happen only a finite number of times accoring to Proposition 2.

If case (ii) or case (iii) happens and there is no player $j \in O_{t'}(\mathbf{d}')$ that wants to deviate, the MB algorithm allows players from the other time slots $t \in \mathcal{T} \setminus \{t'\}$ to perform best improvement steps using *SelfImposedDeviator* (SID) algorithm shown in Fig. 5. Observe that players from time slots $t \in \mathcal{T} \setminus \{t'\}$ are not poked to deviate by the other players, and only reason why they would have an incentive to deviate is that $n_{(t,a)}(\mathbf{d}') < n_{(t,a)}(\mathbf{d}^*(N-1))$ holds for some strategies $(t,a) \in \mathcal{T} \times \mathcal{A}$. The SID algorithm first allows one of the players $j \in O(\mathbf{d}') \setminus O_{t'}(\mathbf{d}')$ that already offloads to perform a best improvement step, and if there is no such player the SID algorithm allows one of the players $j \in L(\mathbf{d}')$ that performs computation locally to start to offload. Let us assume that there is a strategy (t,a) for which $n_{(t,a)}(\mathbf{d}') < n_{(t,a)}(\mathbf{d}^*(N-1))$ holds and that there is a player $j \in O(\mathbf{d}') \setminus O_{t'}(\mathbf{d}') \cup L(\mathbf{d}')$ that wants to deviate to strategy (t,a). We denote by $\mathbf{d}$ the resulting strategy profile, after player j performs its best improvement step. Since $n_{(t,a)}(\mathbf{d}) = n_{(t,a)}(\mathbf{d}') + 1$ and $n_t(\mathbf{d}) = n_t(\mathbf{d}') + 1$ hold, some players $j \in O_{(t,a)}(\mathbf{d})$ may want to perform a best improvement step using the DPD algorithm, which can happen only a finite number of times accoring to Proposition 2. Finally, let us consider case (iii) such that there is a player $j \in O_{t'}(\mathbf{d}') \setminus O_{(t',b)}(\mathbf{d}')$ that wants to stop to offload because its cloud computing cost increased. Let us denote by $\mathbf{d}$ a strategy profile after player j changes its strategy from $(t',a) \neq (t',b)$ to local computing. We have that $n_{(t',a)}(\mathbf{d}) = n_{(t',a)}(\mathbf{d}') - 1$, and if $n_{(t',a)}(\mathbf{d}') = n_{(t',a)}(\mathbf{d}^*(N-1))$ we have that players $j' \in O(\mathbf{d}) \setminus O_{(t',a)}(\mathbf{d})$ may have an incentive to change their offloading strategy to (t',a) if doing so decreases their offloading cost. We have seen that a best improvement step of this type can trigger the DPD algorithm a finite number of times according to Proposition 2. Now, let us assume that a player $j' \in O_{(t,b)}(\mathbf{d})$, where $(t,b) \in \mathcal{T} \times \mathcal{A} \setminus \{(t',a)\}$, changes its offloading strategy from (t,b) to (t',a), and that by doing so it does not trigger the DPD algorithm. The resulting strategy profile $\mathbf{d} = ((t',a), d_{-j'})$ is such that $n_{(t,b)}(\mathbf{d}) = n_{(t,b)}(\mathbf{d}') - 1$ holds, and if $n_{(t,b)}(\mathbf{d}') = n_{(t,b)}(\mathbf{d}^*(N-1))$ some players may have an incentive to change their offloading strategy to (t,b) if doing so decreases their offloading cost.

$(\mathbf{d}, (t,a), A') = SID(\mathbf{d}, A')$

```
/*Players that offload and can decrease their offloading cost*/
D1={j∈O(d)|(t,a)=arg min_{d∈D_j} C_j(d,d_{-j})∈A', d_j≠(t,a)}
/*Players that compute locally and want to start to offload*/
D2={j∈L(d)|(t,a)=arg min_{d∈D_j} C_j(d,d_{-j}) ∈ A'}
if |D1 ∪ D2| ≠ ∅ then
  /*Players that offload have priority*/
  if D1 ≠ ∅ then
    Take i ∈ D1
  else if D2 ≠ ∅ then
    Take i ∈ D2
  end if
  d'_i = arg min_{d∈D_i} C_i(d, d_{-i})
  Let d ← (d'_i, d_{-i})
  Let (t,a) ← d'_i
  Update A'
end if
return (d, (t,a), A')
```

Fig. 5. Pseudo code of the SID algorithm.

We continue by considering the case where all subsequent best improvement steps are such that deviators change to a strategy for which the number of offloaders is less than the number of offloaders in the NE $\mathbf{d}^*(N-1)$ and by doing so they do not trigger the DPD algorithm. Therefore, the resulting best improvement path is such that the cost of each deviator decreases with every new best improvement step it makes. Assume now that after $k \geq 2$ improvement steps player j' wants to return back to strategy (t,b). By the definition of the resulting best improvement path, the cost of player j' in the $(k+1)$-th improvement step is not only less than the cost in the k-th best improvement step, but also less than its cost in the first best improvement step. Therefore, player j' will not return to a strategy it deviated from, and thus it will deviate at most $T \times A - 1$ times. Consequently, when there are no players that can trigger the DPD algorithm, players that change their startegy from local computing to offloading using the SID algorithm, can only decrease their offloading cost in the subsequent best improvement steps, and thus they would have no incentive to stop to offload. Since the number of players is finite, the players will stop changing from local computing to offloading eventually, which proves the theorem. □

Even though the convergence proof of the MB algorithm is fairly involved, the algorithm itself is computationally efficient, as we show next.

Theorem 5. *When a new player i enters the game in an equilibrium $\mathbf{d}^*(N-1)$, the MB algorithm computes a new equilibrium $\mathbf{d}^*(N)$ after at most $N \times T \times A - 2$ best improvement steps.*

Proof. In the worst case scenario the DPD algorithm generates an $N-2$ steps long best improvement path, and a player that offloads in the same time slot as the last deviator, but not through the same AP changes to local computing, because its cloud computing cost increased. Observe that the worst case scenario can happen only if $|O(\mathbf{d}^*(N-1))| = N-1$ holds. Furthermore, $N-2$ players will have an opportunity to deviate using the DPD

algorithm and a player that offloads in the same time slot as the last deviator will have an opportunity to stop to offload only if $n_{(t,a)}(\mathbf{d}^*(N-1)) = n_{(t',b)}(\mathbf{d}^*(N-1))$ holds for every $(t,a),(t',b) \in \mathcal{T} \times \mathcal{A}$. Furthermore, in the worst case scenario, the best improvement path generated by the DPD algorithm is followed by an $N \times (T \times A - 1)$ long best improvement path, in which deviators change to a strategy for which the number of offloaders is less than the number of offloaders in the NE $\mathbf{d}^*(N-1)$ and by doing so they do not trigger the DPD algorithm. Therefore, a NE can be computed in at most $N - 2 + N \times (T \times A - 1)$ best improvement steps. □

By addding players one at a time, it follows that the MB algorithm has quadratic worst case complexity.

Theorem 6. *The MB algorithm computes a NE allocation in $O(N^2 \times T \times A)$ time.*

Implementation considerations: The MB algorithm can be implemented in a decentralized manner, by letting devices perform the best improvement steps one at a time. For computing a best response, besides its local parameters (e.g. D_i, L_i, F_i^0), each device i requires information about achievable uplink rates, available MEC resources, and the number of users sharing the APs and the cloud. In practice these information can be provided by the MEC. As discussed in [5], [18], [7], two main advantages of such a decentralized implementation compared to a centralized one are that the MEC can be relieved from complex centralized management, and devices do not need to reveal their parameters, but only their most recent decisions.

IV. Numerical Results

In the following we show simulation results to evaluate the cost performance and the computational efficiency of the MB algorithm. We consider that the devices are placed uniformly at random over a square area of $1km \times 1km$, while the APs are placed at random on a *regular grid* with A^2 points defined over the area. We consider that the channel gain of device i to AP a is proportional to $d_{i,a}^{-\alpha}$, where $d_{i,a}$ is the distance between device i and AP a, and α is the path loss exponent, which we set to 4 according to the path loss model in urban and suburban areas [19]. For simplicity we assign a bandwidth of 5 *MHz* to every AP a, and the data transmit power of $P_{i,a}$ is drawn from a continuous uniform distribution on $[0.05, 0.18]$ *W* according to measurements reported in [20]. We consider that the uplink rate of a device connected to an AP a scales directly proportional with the number of devices offloading through AP a. The computational capability F_i^0 of device i is drawn from a continuous uniform distribution on $[0.5, 1]$ *GHz*, while the computation capability of the cloud is $F^c = 100$ *GHz* [21]. We consider that the computational capability that a device receives from the cloud scales inversely proportional with the number of devices that offload. The input data size D_i and the number L_i of CPU cycles required to perform the computation are uniformly distributed on $[0.42, 2]$ *Mb* and $[0.1, 0.8]$ *Gcycles*, respectively. The consumed energy per CPU cycle v_i is set to $10^{-11}(F_i^0)^2$ according to measurements reported in [10], [9]. The weights attributed to energy consumption γ_i^E and the response time γ_i^T are drawn from a continuous uniform distribution on $[0, 1]$.

We use three algorithms as a basis for comparison for the proposed *MB* algorithm. In the first algorithm players choose a time slot at random, and implement an equilibrium allocation within their chosen time slots. We refer to this algorithm as the *RandomSlot* (RS) algorithm. The second algorithm considers that all devices perform local execution. The third algorithm is a worst case scenario where all devices choose the same time slot and implement an equilibrium allocation within that time slot. Observe that this corresponds to $T = 1$. We define the *performance gain* of an algorithm as the ratio between the system cost reached when all devices perform local execution and the system cost reached by the algorithm. The results shown are the averages of 100 simulations, together with 95% confidence intervals.

A. Performance gain vs number of devices

Fig. 6 shows the *performance gain* as a function of the number N of devices for $A = 4$ APs. The results show that the *performance gain* decreases with the number of devices for the MB algorithm for all values of T, for the RS algorithm and for the deterministic worst case $T = 1$. This is due to that the APs and the cloud get congested as the number of devices increases. The performance gain of the MB algorithm is up to 50% higher than that of the RS algorithm for $T > 1$; the gap between the two algorithms is largest when the ratio N/T is approximately equal to 4. The reason is that as T increases the average number of offloaders per time slot remains balanced in the case of the MB algorithm. On the contrary, in the case of the RS algorithm some time slots may be more congested than others, since the players choose their time slot at random. However, the average imbalance in the number of offloaders per time slot decreases as the number of devices increases, thus the results are similar for large values of N. At the same time, the performance gain of the MB algorithm compared to that of the deterministic worst case $T = 1$ is almost proportional to the number T of time slots, and shows that coordination is essential for preventing severe performance degradation. It is also interesting to note that for $T = 1$ the *performance gain* decreases with N at a much higher rate than for $T > 1$, which is due to the fast decrease of the number of offloaders, as we show next.

Fig. 7 shows the ratio of players that offload for the same set of parameters as in Fig. 6. The results show that in the worst case, for $T = 1$, the ratio of players that offload decreases almost linearly with N, which explains the fast decrease of the *performance gain* observed in Fig. 6. On the contrary, for larger values of T the ratio of players that offload appears less sensitive to N. We observe that the ratio of players that offload is in general higher in equilibrium than in the strategy profile computed by the RS algorithm, which explains the superior performance of MB observed in Fig. 6.

B. Performance gain vs number of APs

Fig. 8 shows the *performance gain* as a function of the number A of APs for $N = 50$ devices. We observe that the *performance gain* achieved by the algorithms increases

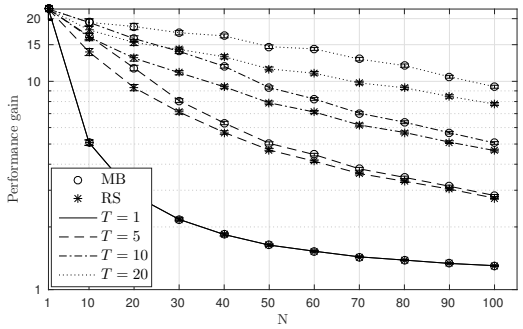


Fig. 6. *Performance gain* vs number of devices (N).

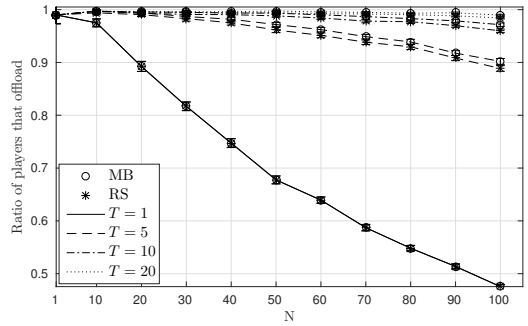


Fig. 7. Ratio of offloaders vs. number of devices (N).

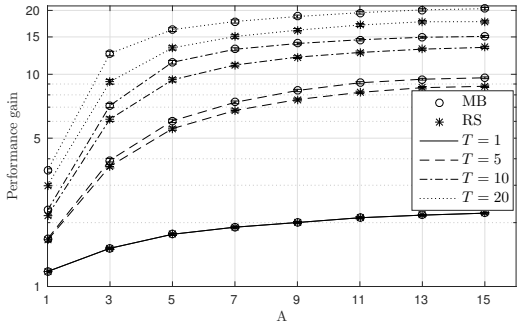


Fig. 8. *Performance gain* vs number of APs (A).

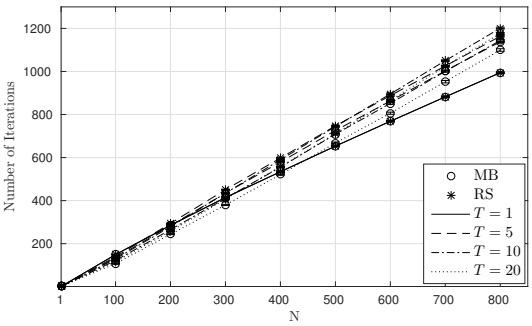


Fig. 9. Number of iterations vs number of devices (N).

monotonically with the number of APs for all values of T with a decreasing marginal gain. The reason is that once $T \times A \geq N$ every device can offload its task through its favorite AP without sharing it, and hence the largest part of the offloading cost comes from the computing cost in the cloud. However, a small change in the *performance gain* is still present even for very large values of A because the density of the APs over a region becomes larger as A increases, and hence the channel gain, which depends on the distance between the device and the APs becomes larger on average. The results also show that MB always outperforms RS, and its *performance gain* compared to that of RS increases with T. Most importantly, the number of APs required for a certain performance gain is almost 50% lower using the MB algorithm compared to the RS algorithm for higher values of T, i.e., significant savings can be achieved in terms of infrastructural investments.

C. Computational Complexity

In order to assess the computational efficiency of the MB algorithm we consider the number of iterations, defined as the number of induction steps plus the total number of update steps over all induction steps needed to compute a NE. Fig. 9 shows the number of iterations as a function of the number N of devices for $A = 4$ APs. The results show that the number of iterations scales approximately linearly with N for both algorithms, and indicates that the worst case scenario considered in Theorem 6 is unlikely to happen. The first interesting feature of Fig. 9 is that the number of iterations is slightly less in the case of the MB algorithm than in the case of the RS algorithm for all values of T, except for $T = 1$ for which the two algorithms are equivalent. The reason is that in the case of the MB algorithm the number of offloaders per time slot is more balanced, and hence the devices have less incentive to deviate when a new device enters the system, and their updates are always at least as good as in the case of RS algorithm, since the MB algorithm allows devices to change between time slots. On the contrary, in the case of the RS algorithm some of the time slots may be very congested, and the devices that offload within these time slots have a higher incentive to deviate when a new device enters the system. The second interesting feature of Fig. 9 is that the number of iterations is smaller for larger values of T for smaller values of N, but for larger values of N the results are reversed. The reason is that for smaller values of N the time slots are less congested on average as T increases, and hence the devices do not want to update their strategies so often. On the contrary, as N increases the benefit of large values of T becomes smaller, because the congestion per time slots increases, and hence devices may want to update their strategies more often.

Overall, our results show that the proposed MB algorithm can compute efficient allocations for periodic task offloading at low computational complexity.

V. Related Work

The scheduling of periodic tasks received significant attention for real-time systems [22], [23], but without considering communications. Similarily, the scheduling of communication resources has been considered without considering computation [24]. Most works that considered both communication and computation considered a single device [25], [10], [6], [26], [27], and thus they do not consider the allocation of resources between devices.

Related to our work are recent works on energy efficient computation offloading for multiple mobile users [28], [29], [30]. [28] proposed a genetic algorithm for maximizing the throughput in a partitioning problem for mobile data stream applications, while [29] proposed a heuristic for minimizing the users' cost in a two-tiered cloud infrastructure with user mobility in a location-time workflow framework. [30] considered minimizing mobile users' energy consumption by joint allocation of wireless and cloud resources, and proposed an iterative algorithm.

A few recent works provided a game theoretic treatment of the mobile computation offloading problem for a

single time slot [31], [32], [5], [18], [33], [34], [7]. [31] considers a two-stage game, where first each mobile user chooses the parts of its task to offload, and then the cloud allocates computational resources to the offloaded parts. [32] considered a three-tier cloud architecture, and provided a distributed algorithm for the computing a mixed strategy equilibrium. [33] considered tasks that arrive simultaneously and a single wireless link, and showed the existence of equilibria when all mobile users have the same delay budget. [5] showed that assuming a single wireless link and link rates determined by the Shannon capacity of an interference channel, the resulting game is a potential game. [18] extended the model to multiple wireless links and showed that the game is still a potential game under the assumption that a mobile user experiences the same channel gain for all links. [7] considered multiple wireless links, equal bandwidth sharing and a non-elastic cloud, and provided a polynomical time algorithm for computing equilibria. Compared to these works, our model of periodic tasks considers the scheduling of tasks over time slots and wireless resources, and is thus a first step towards bridging the gap between early works on scheduling [23] and recent works on computation offloading [5], [7].

From a game theoretical perspective the importance of our contribution is the analysis of a player-specific network congestion game for which the existence of equilibria is not known in general [16], thus the proposed algorithm and our proof of existence advance the state of the art in the study of equilibria in network congestion games.

VI. Conclusion

We provided a game theoretic treatment of computation offloading for periodic tasks. We proved the existence of equilibrium allocations, characterized their structure and provided a polynomial time decentralized algorithm for computing equilibria. Simulations show that the proposed algorithm achieves good system performance for a wide range of system sizes and task periodicities. Our results show that periodic computation offloading can be efficiently coordinated using low complexity algorithms despite the vast solution space and the combinatorial nature of the problem. An interesting open question is whether our results can be extended to devices with heterogeneous periodicities, we leave this question subject of future work.

References

[1] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline, "Pipeneta wireless sensor network for pipeline monitoring," in *Proc. of IPSN*, 2007, pp. 264–273.

[2] S. Oh, P. Chen, M. Manzo, and S. Sastry, "Instrumenting wireless sensor networks for real-time surveillance," in *Proc. IEEE ICRA*, May 2006, pp. 3128–3133.

[3] X. Zhu, S. Han, P. C. Huang, A. K. Mok, and D. Chen, "Mbstar: A real-time communication protocol for wireless body area networks," in *Proc. of ERCTS*, Jul. 2011, pp. 57–66.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing: A key technology towards 5G," Sep. 2015.

[5] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *Proc. of IEEE PDS*, pp. 974–983, 2015.

[6] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing," in *Proc. of IEEE INFOCOM*, April 2013, pp. 1285–1293.

[7] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. of IEEE INFOCOM*, May 2017.

[8] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with pace," in *ACM SIGMETRICS Perf. Eval. Rev.*, vol. 29, no. 1, 2001, pp. 50–61.

[9] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. of Usenix HotCloud*, 2010.

[10] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. of IEEE INFOCOM*, March 2012, pp. 2716–2720.

[11] T. Joshi, A. Mukherjee, Y. Yoo, and D. P. Agrawal, "Airtime fairness for ieee 802.11 multirate networks," *IEEE Trans. on Mobile Computing*, vol. 7, no. 4, pp. 513–527, 2008.

[12] C. U. Saraydar, N. B. Mandayam, and D. J. Goodman, "Efficient power control via pricing in wireless data networks," *IEEE Trans. on Communications*, vol. 50, no. 2, pp. 291–303, 2002.

[13] M. Xiao, N. B. Shroff, and E. K. Chong, "A utility-based power-control scheme in wireless cellular systems," *IEEE/ACM Trans. on Networking*, vol. 11, no. 2, pp. 210–221, 2003.

[14] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, Jun. 2012.

[15] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer Mag.*, vol. 43, no. 4, pp. 51–56, Apr. 2010.

[16] I. Milchtaich, "The equilibrium existence problem in finite network congestion games," in *Proc. of WINE*, 2006, pp. 87–98.

[17] ——, "Congestion games with player-specific payoff functions," *Games and Economic Behavior*, vol. 13, no. 1, pp. 111 – 124, 1996.

[18] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[19] A. Aragon-Zavala, *Antennas and propagation for wireless communication systems*. John Wiley & Sons, 2008.

[20] E. Casilari, J. M. Cano-García, and G. Campos-Garrido, "Modeling of current consumption in 802.15. 4/zigbee sensor motes," *Sensors*, vol. 10, no. 6, pp. 5443–5468, 2010.

[21] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *ISCC*, 2012, pp. 59–66.

[22] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. on Computers*, vol. 39, pp. 1175–1185, Sep. 1990.

[23] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Syst.*, vol. 28, no. 2-3, pp. 101–155, Nov. 2004.

[24] I. H. Hou, "Packet scheduling for real-time surveillance in multihop wireless sensor networks with lossy channels," *IEEE Trans. on Wireless Comm.*, vol. 14, no. 2, pp. 1071–1079, Feb 2015.

[25] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proc. of ACM MobiSys*, 2010, pp. 49–62.

[26] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mob. Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Feb 2013.

[27] E. Hyytiä, T. Spyropoulos, and J. Ott, "Offload (only) the right jobs: Robust offloading using the Markov decision processes," in *Proc. of IEEE WoWMoM*, Jun. 2015, pp. 1–9.

[28] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, Apr. 2013.

[29] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-aware optimal service allocation in mobile cloud computing," in *Proc. of IEEE CLOUD*, Jun. 2013, pp. 75–82.

[30] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE T-SIPN*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[31] Y. Wang, X. Lin, and M. Pedram, "A nested two stage game-based optimization framework in mobile cloud computing system," in *Proc. of IEEE SOSE*, Mar. 2013, pp. 494–502.

[32] V. Cardellini et al., "A game-theoretic approach to computation offloading in mobile cloud computing," *Mathematical Programming*, pp. 1–29, 2015.

[33] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy efficient offloading for competing users on a shared communication channel," in *Proc. of IEEE ICC*, Jun. 2015, pp. 3192–3197.

[34] X. Ma, C. Lin, X. Xiang, and C. Chen, "Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing," in *Proc. of ACM MSWiM*, 2015, pp. 271–278.

Experience-Availability Analysis of Online Cloud Services using Stochastic Models

Yue Cao[1], Laiping Zhao[1*], Rongqi Zhang[2], Yanan Yang[1], Xiaobo Zhou[2], Keqiu Li[2]
[1]*School of Computer Software, Tianjin University, Tianjin, China*
[2]*Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, Tianjin, China*
E-mail: {cy1994, laiping, zrq0312, ynyang, xiaobo.zhou, keqiu}@tju.edu.cn

***Abstract*—Recently a new performance metric called experience availability (EA) has been proposed to evaluate online cloud service in terms of both availability and response time. EA originates from the fact that from the prospective of quality of experience (QoE), an online cloud service is regarded as unavailable not only when it is inaccessible, but also when the tail latency is high. However, there still lacks analytic models for evaluating the EA of online services. In this paper, we propose an efficient EA-analytic model using stochastic reward net (SRN) to study the tail latency performance of online cloud services in the presence of failure-repair of the resources. Our EA-analytic model can predict the online service performance on EA, as well as support analysis on traditional availability and mean response time. We apply this model to an Apache Solr search service, and evaluate the prediction accuracy by comparing the results derived from the model to actual experimental results. It is shown that the proposed model overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. Through attribution analysis, we further identify the list of factors that may affect the accuracy, and show that the 95^{th} percentile latency prediction error can be reduced to as low as 2.45% by tuning the configurations suggested by the attribution.**

***Keywords*—cloud computing, experience availability, online cloud service, stochastic reward net**

I. Introduction

Cloud computing is growing rapidly towards delivering computing as a public utility. Many services, including online web systems (e.g., social networking, e-commerce, search engine) and offline data-processing jobs (e.g., mapreduce, spark), are continuously deployed or processed in cloud systems [1, 2]. These services often consist of multiple tiers and tens or hundreds of tasks or micro-services, and need to handle unprecedented volumes of data. To characterize the performance of cloud service, an uptime-based *availability* measure was widely used [3], which is defined as

$$A = \frac{MTTF}{MTTF + MTTR}, \tag{1}$$

where MTTF and MTTR denote mean time to failure and mean time to repair, respectively. According to Equ. (1), availability describes only whether the service is accessible or not.

* Corresponding Author: Laiping Zhao, laiping@tju.edu.cn

In fact, from the prospective of quality of experience (QoE), the tail latency performance is at least as important as availability for a realtime online cloud service. Google's experiences on their back end services show that while majority of requests take around 50-60 ms, a fraction of requests takes longer than 100 ms, with the largest difference being almost 600 times [4]. One major reason of the performance uncertainty is due to the inevitable underlying competition on hardware resources among co-located services, resulting in the serious tail latency problem [5]. According to Nielsen [6], 0.1 second is about the limit for having the user feel that the system is reacting instantaneously; a response time of 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. For delays longer than 10 seconds, users are prone to perform other tasks while waiting. In this sense, *slow response* and *service unavailable* would be indistinguishable for cloud users [7].

Recently a new performance measure, named *Experience Availability* (EA), which is extended from the definition of *availability*, is proposed in [8]. It combines the traditional availability and tail latency for the first time into a single metric. According to EA, a service is *experience unavailable* not only because the service is failed, but also the γ^{th} latency exceeds a pre-defined threshold τ.

Currently, there still lacks analytic models and methodologies to analyze the EA of cloud services. According to its definition, we need to derive both the tail latency and availability to measure EA. Analyzing availability is relatively straightforward, and there are already a number of models proposed for availability modeling [9, 10]. However, calculating tail latency of online cloud services is a significant challenge due to the multi-tier architecture of cloud services. The errors could propagate across tiers and have cascading effects on overall latency distribution. Even worse, the majority of existing work focus on evaluating mean performance metrics, such as average response time, average resource utilization [11], while there are only a few of them considering to model and analyze the distribution of latencies for online services.

In this paper, we propose an efficient analytic model for analyzing the EA of online cloud services. We consider the common online search service, and use a stochastic reward net (SRN) to describe the interactions between its multi-tiers.

ISBN 978-3-903176-08-9

Through the model analysis, we can predict the service performance on EA. By comparing the prediction results to the real experimental results, we find that the proposed model shows some error on tail latency. We then give an attribution analysis on the system behavior, and find the potential factors that may affect the accuracy. The contributions can be summarized as follows:

- We design a SRN model to characterize the request response process of online cloud services. In this model, we study the mean response time of online cloud services in the presence of failure-repair of the resources.
- We propose a tagged customer model to analyze the cumulative distribution function (CDF) of response time for online cloud services. Based on the CDF, we can derive the EA of online cloud services.
- We demonstrate the accuracy of the model by comparing the analytical results to the real experimental results. It shows that the proposed model overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. We further conduct experiments to identify a list of factors, including cache, number of keywords in search space, turbo boost and DVFS governor, that may affect the accuracy. It is found that by turning off the cache, increasing the search space, turning off Turbo Boost and configuring the DVFS performance governor, the prediction error can be reduced to as low as 2.45%.

The rest of the paper is organized as follows. Section II describes the basic architecture of online search service and our design of the SRN model. We present the EA-analytic model and introduce the model solving method in section III. Section IV describes the experimental evaluation of our model for online search service. In section V, we summarize the related work. We conclude and discuss future work in section VI.

II. Search Service and the SRN Model

In this section, we firstly introduce the general architecture of online search service. Then, we show how to construct the SRN model for the online search service.

A. Online Service Architecture

An online search service is a software system that is designed to search for information on the World Wide Web. Generally it consists of three main components [12]: the web crawler, the index generator, and the search engine, as shown in Fig. 1. The web crawler crawls some of the reachable web pages from site to site. The index generator associates keywords found on these web pages to their names of sites containing the keywords. It uses an update handler to process all updates, and generates distributed indexes. The indexed information is stored in database, and made available for search queries. The search engine will accept user's search requests, support text analysis, and generate the web pages list results by searching indexes. During this process, every page in the entire list must be weighted according to information in the indexes.

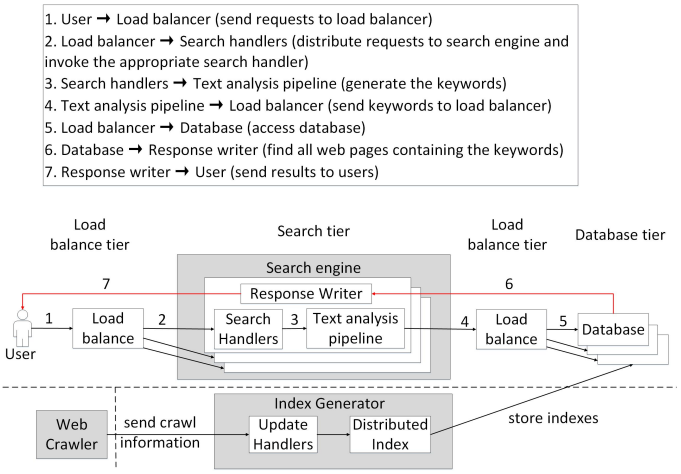


Fig. 1. General architecture of the online search service.

Since a user's perception of the latency is mainly affected by the search engine, we hereafter merely discuss the modeling of the request processing in search engine. As shown in Fig. 1, a typical infrastructure for supporting a search component mainly includes four tiers: search tier, database tier and two load balancer tiers. We do not take into account the two load balancer tiers because their processing time is negligible compared to the search tier and the database tier in our setup. Depending on the number of users, each tier can be implemented with multiple instances, which are hosted across different virtual machines (VMs) in cloud platform. If there exists more than one instance at a tier, it is required to deploy another load balance tier to distribute the customer requests among instances.

Each instance, either the search or the database, maintains a thread pool for accepting requests. When a search request arrives, it first waits in the searching queue. The load balancer reads the search queue, and dispatches the request to a specific instance of search tier for text analysis. The corresponding instance then allocates one connection thread from thread pool to the request, and the connection will be occupied until user receives search results. After getting the keywords by text analysis, the request is further forwarded to an instance of database. Then, the database instance also allocates a thread to it for searching all web pages containing the keywords. Then, the response writer in the search instance constructs a ranked list of web pages, and sends the results back to user.

B. SRN Model

According to the architecture of online search service shown in Fig. 1, we construct a SRN model to analyze the interactions between multi-tiers. SRN is scalable to model systems composed of thousands of resources and is flexible to represent different policies and strategies [13].

We assume that the times assigned to all timed transitions conforms to an exponential distribution between, following

the common assumptions in [11, 14]. We consider crash failures occurs in an instance, i.e., an instance of search or database could fail with probabilities, resulting in the lost of all connections running on the instance. If a request's connection is lost due to the instance failures, we think that its response time is infinity. The input parameters required in the SRN model include: (1) request arrival rate (denoted by λ); (2) queue sizes of search and database (denoted by M_s and M_{db}); (3) service rates of a search instance or database instance (denoted by μ_s and μ_{db}); (4) the maximum number of connections supported by search tier and database tier (denoted by N_s and N_{db}); (5) failure rates of search and database (denoted by ϕ_s and ϕ_{db}); (6) repair rates of search and database (denoted by δ_s and δ_{db}).

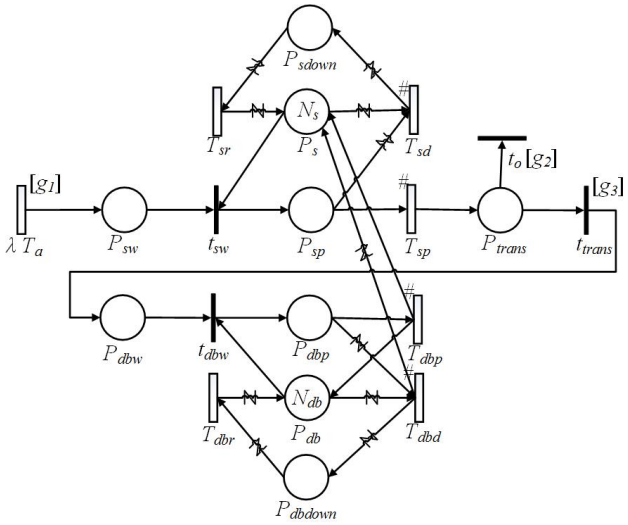


Fig. 2. SRN model of the online search service.

TABLE I
GUARD FUNCTIONS OF THE SRN MODEL

Transition	Guard Function
g1	$\#P_{sw} < M_s$? 1 : 0
g2	$\#P_{dbw} \geq M_{db}$? 1: 0
g3	$\#P_{dbw} < M_{db}$? 1 : 0

Fig. 2 describes the design of the SRN model for online search service. We only consider the two main tiers that directly affect the user's response time in the model: place P_{sw} to transition T_{sp} represents the processing in the search tier; place P_{dbw} to transition T_{dbp} represents processing in the database tier. Components such as search handlers are not studied separately because their processings have been incorporated into their corresponding tiers. Timed transition T_a represents the request arrivals to the system. Places P_{sw} and P_{dbw} represent the waiting queues of the search tier and database tier, and T_a fires only if the queue of search tier is not yet full (following guard function g_1). Once transition T_a (t_{trans}) fires, a token is deposited in place P_{sw} (P_{dbw}) showing that a request has been submitted to search tier (database tier) and it is waiting for processing from search tier (database tier). Place P_s and place P_{db} represent the remaining number of resources (i.e., connection threads) supported by search and database tier, and they are initialized with N_s and N_{db}. If there is a token in place P_{sw} and there is at least one token in place P_s, then one token from P_{sw} together with another token from P_s is removed respectively, and a token is put in place P_{sp}, representing that a request is ready for processing by search tier. Places P_{sp} and P_{dbp} represent the processing queues of search tier and database tier. The pound # in the arc from place P_{sp} to the transition T_{sp} shows that the actual firing rate of transition is marking-dependent. Thus, the actual firing rate of transition T_{sp} is calculated as $K\mu_s$, where K is the number of tokens in place P_{sp}. After that, a token is removed from place P_{sp} and deposited into place P_{trans}, which represents that the request has been processed and leaves from search tier. Transition t_{trans} represents that the request moves from search tier to database tier. Transition t_o shows that the request gets dropped from the system and it fires only when the queue of database tier is already full (following guard function g_2). And then, the request is inserted into the waiting queue of database (i.e., place P_{dbw}) following guard function g_3. At the database tier, likewise, the request is processed only if database has available resources. After firing the timed transition T_{dbp}, a token is removed from place P_{dbp}, while one token deposited into place P_{db} and another token deposited into place P_s, showing that the request is finished and the corresponding connections at search tier and database tier are both released.

In the SRN model, we also model and analyze the impact on response time by server failures. We use place P_{sdown} and P_{dbdown} to represent the number of lost connections in search tier and database tier, respectively. If an search instance fails, all working connections in P_{sp} and idle connections in P_s running on the instance are lost simultaneously. Transition T_{sd} represents the failure occurs in search instances. The zigzag line on an arc represents that the arc can transfer multiple tokens at once. Suppose there are I_s instances at search tier and I_{db} instances at database tier. Since the requests are distributed evenly among the instances, a fraction of $1/I_s$ tokens in place P_{sp} together with another fraction of $1/I_s$ tokens in place P_s are removed respectively, and the sum of these tokens (N_s/I_s) is put in place P_{sdown}, representing that failed connections are ready for recovering. Once transition T_{sr} fires, the number of tokens (N_s/I_s) are removed from P_{sdown} and deposited in place P_s, indicating that the lost connections are recovered. The same process also applies to the database tier, except that the fraction of $1/I_{db}$ tokens are also deposited in place P_s after transition T_{dbd} fires. It means that the affected connections at search tier are also released when a database instance fails. The guard functions of the SRN model is shown in Tab. I.

C. Mean Response Time under Steady State

Given the SRN model above, we can derive the mean response time characterizing the system behavior by defining reward functions. To analyze the mean response time, we

first need to derive the mean number of waiting requests and blocking probability of requests.

1) Mean number of waiting requests: The mean number of waiting requests is given by mean number of tokens in place P_{sw} in Fig. 2, and it can be represented by $E[\#P_{sw}]$.

2) Blocking probability of requests: The steady-state blocking probabilities of requests in Fig. 2, P_b, can be calculated by assigning the following reward to the SRN model,

$$r_i = \begin{cases} 1, if[\#P_{sw}] \geqslant M_s, \\ 0, \text{otherwise.} \end{cases} \tag{2}$$

where M_s is the maximum length of waiting queue.

3) Mean response time: Mean response time is defined as the mean time from a customer request is entered the system until its leave. According to Fig. 2, the mean response time is the time from the instant that a token is deposited into P_{sw} until it is removed from P_{dbp}. Using Little's law, the mean response time (denoted by E_t) for requests at steady-state can be calculated as follows,

$$E_t = \frac{(E[\#P_{sw}] + E[\#P_{dbw}] + E[\#P_{sp}] + E[\#P_{dbp}])}{(1 - P_b) \times \lambda} \tag{3}$$

where $E[\#P_x]$ is the mean number of tokens in place P_x at steady-state and (1 - P_b) $\times$ λ is the effective request arrival rate in the online search service system.

III. Experience Availability Measure

In this section, we show how to model and predict the EA of the online search service. Generally, it takes three steps for calculating EA: (1) Divide the total operational time *T* into *n* time slices. (2) In each time slice, derive the CDF of response time while taking into account instance failures. (3) Derive EA based on the CDF of response times in all time slices.

A. Tagged Customer Model

To derive the CDF of response time, we propose the tagged customer model [15] by modifying the SRN model in order to track the tagged customers movements through the system.

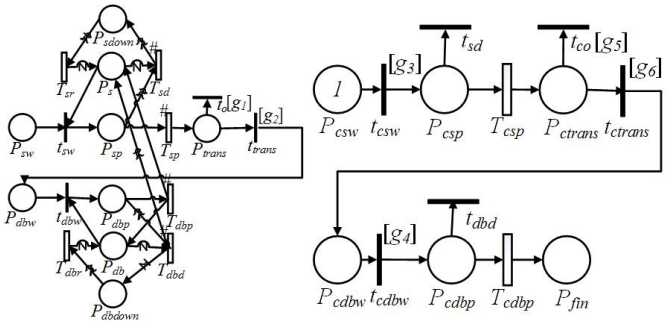


Fig. 3. Tagged customer model.

We show the design of the tagged customer model in Fig. 3. Place P_{csw} contains a single token that represents the arrival of a request. The *m*, *n*, *i*, *j*, *a*, *b* and *p*, *q* tokens initially presented in places P_{sw}, P_{dbw}, P_{sp}, P_{dbp}, P_{sdown}, P_{dbdown} and P_s, P_{db}, represent the corresponding system status before the request arrival. That is, at the time of the request arrival, there are *m* (n) requests waiting in the search (database) queue, i (j) requests being processing by search (database), a (b) connections lost in P_{sdown} (P_{dbdown}), and p (q) connection threads in the thread pool still available for accepting new requests. Transition t_{csw} is fired only when place P_{sw} is empty and the place P_s is not empty. Transition t_{cdbw} is fired only when place P_{dbw} is empty and the place P_{db} is not empty. The guard functions are shown in Tab. II.

TABLE II
Guard Function of the Tagged Customer Model in Fig. 3

Transition	Guard Function
g1	$\#P_{dbw} \geq M_{db}$? 1 : 0
g2	$\#P_{dbw} < M_{db}$? 1 : 0
g3	$\#P_s > 0$ and $\#P_{sw} == 0$? 1 : 0
g4	$\#P_{db} > 0$ and $\#P_{dbw} == 0$? 1 : 0
g5	$\#P_{dbw} \geq M_{db}$? 1 : 0
g6	$\#P_{dbw} < M_{db}$? 1 : 0

B. Response Time Distribution Calculation

Now, we can derive the response time CDF using the tagged customer model. We define the set of initial system states as follows:

$$T = [m, n, i, j, a, b, p, q] \tag{4}$$

where,

$$m \in [0, ..., M_s], n \in [0, ..., M_{db}], i \in [0, ..., N_s], j \in [0, ..., N_{db}],$$
$$a \in [0, ..., N_s], b \in [0, ..., N_{db}], p \in [0, ..., N_s], q \in [0, ..., N_{db}],$$
$$i + a + p = N_s, j + b + q = N_{db}. \tag{5}$$

Denoted by π_x the probability that the system stays in $\forall x \in T$ under steady state. Then π_x can be derived using the SRN model proposed in Section II. Clearly, we have,

$$\sum_{x \in \tau} \pi_x = 1 \tag{6}$$

In the tagged customer model, a non-empty place P_{fin} means that the processing of the tagged request has been completed. Thus, we define the absorbing state for the tagged customer as follows,

$$r_x(t) = \begin{cases} 1, if(\#(P_{fin}), t) > 0, \\ 0, \text{otherwise.} \end{cases} \tag{7}$$

where the reward function $r_x(t)$ is denoted whether an absorbing marking has been reached at time *t*.

By solving the reward function $r_x(t)$, we can obtain the probability $R_x(t)$ that the tagged customer request is absorbed at time t under initial marking x. Then, the probability that a request processing is completed at time t can be calculated as follows:

$$R(t) = \sum_{x \in \tau} [R_x(t) \times \pi_x] \tag{8}$$

Given the $R(t)$ and the pre-defined γ and τ, we can easily know if the γ^{th} percentile latency exceed τ or not in the current time slice. Then, it is straightforward to calculate the EA using Equ. (9),

$$EA(\tau,\gamma) = \frac{\sum_{i=1}^{m_i} t_{TTHTL_i(\tau,\gamma)}}{\sum_{i=1}^{m_i} t_{TTHTL_i(\tau,\gamma)} + \sum_{j=1}^{n_i} t_{TTLTL(\tau,\gamma)}} \quad (9)$$

where $t_{TTHTL_i(\tau,\gamma)}$ represents the time to high tail latency, that is, the time whose γ^{th} percentile latency is less or equal to τ: $TL(\gamma) \leq \tau$, and $t_{TTLTL_j(\tau,\gamma)}$ represents the time to low tail latency, that is, the time with $TL(\gamma) > \tau$.

IV. Experimental Evaluation

In this section, we evaluate the proposed SRN model by comparing its results with the actual experimental results. We then give an in-depth analysis on the prediction error, and list the factors that may affect the accuracy of the model.

A. Experiment Setup

We use the Stochastic Petri Net Package (SPNP) [16] to solve the proposed model. We also implement a real Apache Solr [17] search service to record and analyze the actual tail latency experienced by users. All input parameters to the SRN model are extracted from the testing of the Apache Solr service. The proposed SRN model does not take into account the two load balancers because their processing time is negligible compared to the application and database tiers in our setup.

For the real Apache Solr service, we deploy three instances of SolrCloud search engine, which is implemented using Tomcat for accepting users' search requests, supporting text analysis and interacting with database tier. Each instance of the SolrCloud supports a limited number of connections. When a request arrives at the server, a separate available connection thread will be allocated to the request. Likewise, we also deploy three instances of MySQL database to store the indexes generated by crawler and indexing component, and each database instance is configured with a limited maximum number of connected clients. There may be more than three instances in real system. Our model can still work by simply changing the number of instances in the configurations. However, the runtime of model will increase over the number of instances, and studying the scalability is left as future work. In addition, either the SolrCloud instance or the MySQL instance probably fails following some pre-configured failure rates. To demonstrate the ability of our model, we artificially set a relative high failure rate for both SolrCloud and MySQL instances in the SRN model. We also injected failures into the real system by powering off instances at the same rate. If an instance fails, all connections running on it will be lost simultaneously. The connections will not be recovered until the instance is repaired.

Search requests are automatically generated by Apache Jmeter [18]. It randomly selects a keyword from a pre-configured search space, and sends requests to Solr following a Poisson distribution with the arrival rate λ. The search space is initially configured with 3,000 keywords. We totally send 1 million search requests to Solr service. Then Jmeter records all response times of the requests. We deploy the Apache Solr service on machines configured with Intel Core i5-4670 processor, and 8GB RAM. To estimate the service rate of both search threads and database threads, we use Jmeter to access Solr and Mysql separately, and take the average of ten repeated operations as the final service rate. Tab. III lists all parameter configurations used in our model and actual experiments.

TABLE III
Configuration Parameters

Parameter	Values		
Max # of connections in search tier (N_s)	9	30	45
Max # of connections in database tier (N_{db})	9	30	45
Waiting queue size in search tier (M_s)	30		
Waiting queue size in database tier (M_{db})	30		
Request arrival rate (λ)	100	400	900
Service rate of a search thread (μ_s)	56		
Service rate of a database thread (μ_{db})	32		
Failure rate of a search instance (ϕ_s)	0.100	0.125	0.330
Failure rate of a database instance (ϕ_{db})	0.100	0.125	0.330
Repair rate of a search instance (δ_s)	0.100		
Repair rate of a database instance(δ_{db})	0.100		

B. Results

1) Mean Response Time: We first evaluate the mean response time of the Solr service using the configuration parameters listed in Tab. III.

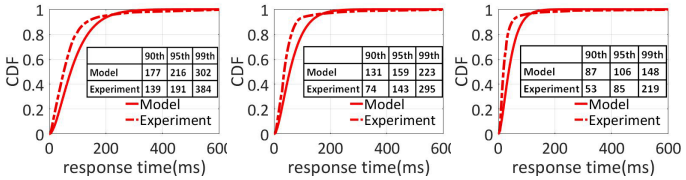
TABLE IV
Mean Response Time

Configurations					Mean response time (ms)		
N_s	N_{db}	λ	ϕ_s	ϕ_{db}	Model	Experiment	Error
9	9	400	0.125	0.125	89.52	74.49	20.18%
30	30	400	0.125	0.125	57.93	49.31	17.48%
45	45	400	0.125	0.125	40.16	33.97	18.22%
30	30	100	0.125	0.125	43.29	37.15	16.53%
30	30	900	0.125	0.125	97.58	82.83	17.81%
30	30	400	0.330	0.330	68.32	57.14	19.57%
30	30	400	0.100	0.100	51.86	43.79	18.43%

Tab. IV shows the SRN model prediction results and the actual experimental results. Fixing the request arrival rate λ at 400 and failure rates ϕ_s and ϕ_{db} at 0.125, the response times of both the SRN model and actual experiments decrease as the number of maximum supported connections increase from 9 to 45. While setting $N_a = N_{db} = 30$, the response time increase significantly as the job arrival rate increase from 100 to 900.

Moreover, the response time decrease if we reduce the failure rate of both search instance and MySQL instance. We see that the relative error on mean response time is generally within 20.18%, and the SRN model results on mean response time are mostly larger than the actual experimental results. This is primarily because, there are some acceleration techniques adopted by either the hardware layer or software layer, for example, CPU pipeline, multiple instruction issue, cache, our SRN model is unable to analyze these techniques, while they indeed reduce the response time significantly in the actual experiments.

2) Response Time CDF: By changing the maximum number of connections (N_s and N_{db}), request arrival rate (λ), and failure rate (ϕ_s and ϕ_{db}), respectively, we conduct three groups of experiments to evaluate the cumulative distribution function (CDF) of the response time.

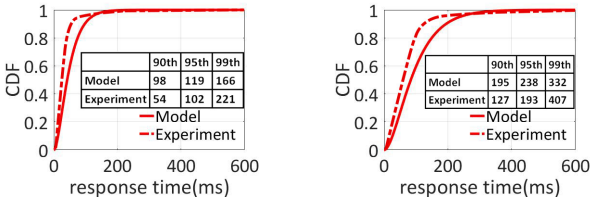


(a) N_s = 9, N_{db} = 9 (b) N_s = 30, N_{db} = 30 (c) N_s = 45, N_{db} = 45

Fig. 4. Response time CDF when changing the max. number of connections.

The first experiment evaluates the response time CDF under settings with different maximum number of connections (N_s and N_{db}). For the other parameters, we set $\lambda = 400$, $\phi_s = \phi_{db} = 0.125$. Fig. 4 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when N_s and N_{db} are set to 9, 30 and 45, respectively. We find that increasing the number of connections would reduce the 90^{th}, 95^{th}, 99^{th} percentile latency in both model analysis and actual experiments. On the one hand, our model overestimates the response time at lower percentile due to the lack of consideration on accelerating technologies such as cache, CPU pipeline, multiple instruction issue etc. On the other hand, our model underestimates the response time at higher percentile (99^{th}). This is because, there are many processes, from either other instances/applications or operating system processes, running on the same underlying hardware simultaneously, yet the hardware does not support performance isolation between these processes, resulting in resource contention and disorder. Hence, the higher percentile latency in actual experimental is usually much larger than model results.

The second experiment evaluates the response time CDF under settings with different request arrival rate (λ). Fig. 5 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when $N_s = N_{db} = 30$ and λ is set to 100 and 900, respectively. We see that the 90^{th}, 95^{th}, 99^{th} percentile latency all increases as we increase the arrival rate. Like the first experiment, our model overestimates the response time at lower percentile, and underestimates the response time at higher percentile (99^{th}). The two lines cross at around the 96^{th} percentile.

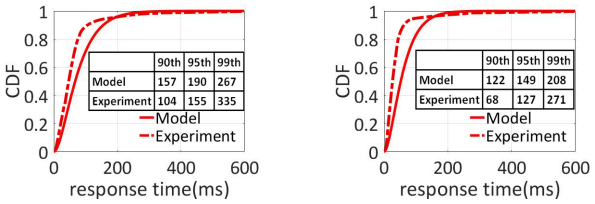


(a) λ = 100 (b) λ= 900

Fig. 5. Response time CDF under different request arrival rates.

TABLE V
EXPERIENCE AVAILABILITY

Configurations					Results			
N_s	N_{db}	λ	ϕ_s	ϕ_{db}	250ms percentile		EA	A
					Model	Experi.		
9	9	400	0.125	0.125	97.3%	96.4%	70%	96.3%
30	30	400	0.125	0.125	99.5%	98.9%	85%	96.5%
45	45	400	0.125	0.125	99.9%	99.9%	95%	97.0%
30	30	100	0.125	0.125	99.9%	99.9%	95%	96.3%
30	30	900	0.125	0.125	95.9%	96.3%	80%	96.2%
30	30	400	0.330	0.330	98.6%	97.4%	75%	90.7%
30	30	400	0.100	0.100	99.7%	99.1%	90%	97.1%



(a) ϕ_s = 0.33, ϕ_{db} = 0.33 (b) ϕ_s = 0.1, ϕ_{db} = 0.1

Fig. 6. Response time CDF under different failure rate.

The third experiment evaluates the response time CDF under settings with different failure rate (ϕ_s and ϕ_{db}). Fig. 6 shows the response time CDF and the 90^{th}, 95^{th}, 99^{th} percentile latency, when ϕ_s and ϕ_{db} are set to 0.1 and 0.33, respectively. When failures are injected to search and database instances, all the affected connections will be lost, and their response times become infinity. Moreover, since the maximum number of connections supported by search tier and database tier is reduced due to the failures, the newly arrived requests more probably wait in the queue, or even are rejected if the waiting queue becomes full. Thus, increasing the failure rate would increase the response time.

TABLE VI
ERROR ATTRIBUTION FACTORS.

Factor	Level 1	Level 2
Cache	off	on
Number of Keywords	8000	3000
Turbo Boost	off	on
DVFS Governor	performance	ondemand

3) Experience Availability Calculation: Suppose the user's QoE requirement on tail latency is defined as: in each time slice, the 99^{th} percentile latency should be less than 250ms. Tab. V shows the percentiles at the response time of 250ms in different configurations, calculated by solving the SRN model and actual experiments, respectively. We see that the percentiles calculated by SRN model is larger than that in the actual experiment. By dividing the 1 million requests into 20 equal-time-intervals, we calculate a statistical analysis of the response times for each time slice, and calculate the EA according to Equ. (9). We find that, even the arrival rates are the same in all time slices, but the response time we measure is changing across runs, resulting in different percentiles at 250ms. This is due to the underlying hardware contention interferences from other processes in the same server. We also calculate the traditional availability according to Equ. (1). Clearly, its values are generally much larger than EA, implying that a service may be experience-unavailable even when it is available.

C. Error Attribution

The analysis has shown the possible prediction error by our model. To reduce the prediction error, we need to identify the potential factors that may affect the accuracy. In this section, we list all the factors we suspect to have an impact on the response time, and evaluate their impact on the response time.

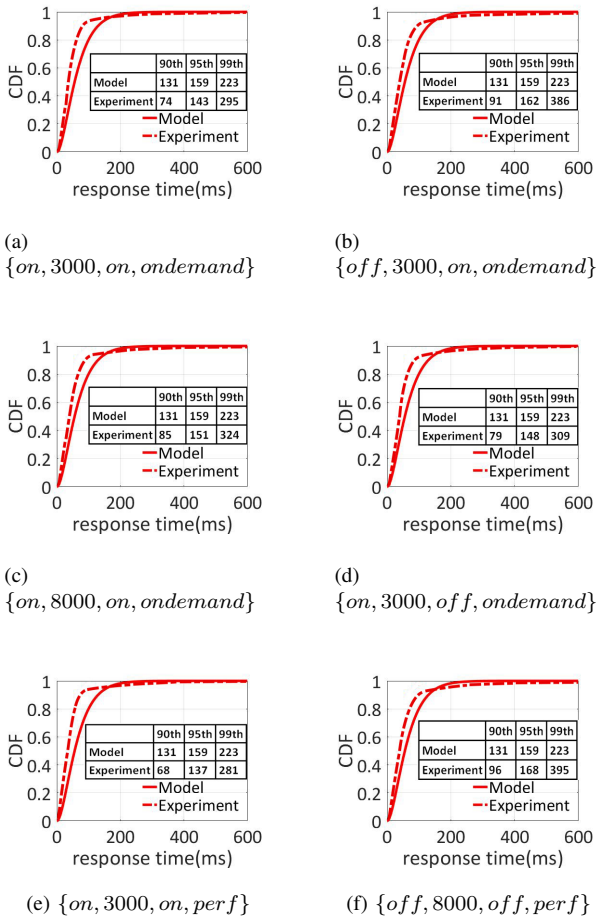
1) Cache: The cache includes CPU cache and page cache. CPU cache is used by the CPU of a computer to reduce the average time to access data from the main memory. The page cache is kept by the operating system to speed-up the access to the contents of cached pages and improve the overall performance.

2) Number of keywords in search space: Since Jmeter randomly selects keywords from the search space in each request, a small search space could lead to high frequent repeated requests. A repeated request could be completed by the cache not only in the service side, but also in the client.

3) Turbo boost: Turbo boost is a feature implemented on many modern processors, which automatically raises the processors' operating frequency, and thus performance, depending on the task demand and dynamic power.

4) DVFS Governor: Dynamic voltage and frequency scaling is a power management technique in CPU. It allows the operating system to dynamically adjust the CPU frequency to boost performance or save power.

Tab. VI lists all possible configurations of the factors, which are divided into two levels, where level 1 indicates {*cache off, 8000 keywords, turbo boost off, DVFS governor performance*}, and level 2 indicates {*cache on, 3000 keywords, turbo boost on, DVFS governor ondemand*}. We conduct another group of experiments to evaluate the impact of these factors on the model prediction accuracy as shown in Fig. 7. We find that, setting all the four factors at level 1 could help to reduce the prediction error on mean response time, 90^{th} and 95^{th} percentile latency. In particular, as shown in Tab. VII, the prediction error on mean response time could be reduced to as low as 7.12%, the prediction error on 95^{th} percentile latency could be reduced to 2.45%. Among the four factors, we find that turning off the cache feature is the most effective way to reduce the error, the next is increasing the number of keywords, turbo boost and DVFS governor make little effect on the results. For the 99^{th} percentile latency, we see that the configuration of level 1 actually increase the prediction error to 43.54%. This is because the interferences coming from hardware resource contention still exists, and we are not able to eliminate their impact through simple configurations.



(a) $\{on, 3000, on, ondemand\}$ (b) $\{off, 3000, on, ondemand\}$

(c) $\{on, 8000, on, ondemand\}$ (d) $\{on, 3000, off, ondemand\}$

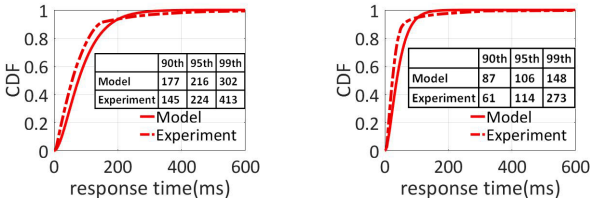
(e) $\{on, 3000, on, perf\}$ (f) $\{off, 8000, off, perf\}$

Fig. 7. Response time CDF under different configurations in Tab. VI.

Fig. 8 shows the response time CDF under the settings of $N_s = N_{db} = 9$ and $N_s = N_{db} = 45$, respectively, after we change all the factors into level 1. Compared with the results in Fig. 4, when $N_s = N_{db} = 9$, the prediction error on mean

TABLE VII
ERROR ANALYSIS CORRESPONDING TO FIG. 7.

Results		Mean Time		Tail Latency (ms)					
				90^{th}		95^{th}		99^{th}	
Model		57.93		131		159		223	
Experiment		Value	Error	Value	Error	Value	Error	Value	Error
	a	49.31	17.48%	74	77.03%	143	11.19%	295	24.41%
	b	53.47	8.34%	91	43.96%	163	2.45%	386	42.23%
	c	52.81	9.70%	85	54.12%	151	5.30%	324	31.17%
	d	50.36	15.03%	79	65.82%	148	7.43%	309	27.83%
	e	48.25	20.06%	68	92.65%	137	16.06%	281	20.64%
	f	54.08	7.12%	96	36.46%	168	5.36%	395	43.54%



(a) N_s = 9, N_{db} = 9 (b) N_s = 45, N_{db} = 45

Fig. 8. Response time CDF for different resource number.

response time is reduced to 9.79%, and the 95^{th} percentile latency is reduced to 3.57 %. When $N_s = N_{db} = 45$, the prediction error on mean response time is reduced to 8.47%, and the 95^{th} percentile latency is reduced to 7.02%.

V. RELATED WORK

A. Availability Evaluation

There has been a large amount of work on evaluating the availability of cloud services, which can be categorized into model based methods and measurement based methods.

In model based methods, three types of widely used models are combinatorial models, state-space models and hierarchical models. Combinatorial models include Reliability Block Diagrams (RBD) and fault tree analysis. RBD are proposed to model the availability of virtual data centers (VDCs) [19] and fault tree [20] has been used to evaluate the reliability of multi-nodes SDDC (software-defined data center). Although it is easy to implement combinatorial model due to its explicit presentation, it cannot model large and complicated systems.

State-space models mainly include Markov chain, semi-Markov processes, stochastic petri net (SPN) or stochastic reward net (SRN) [21] . Wu et al. [22] presented a stochastic method based on semi-Markov model to evaluate the availability of Infrastructure-as-a-Service (IaaS) cloud. Longo et al. [23] presented an SRN model to analyze the availability of a large-scale IaaS cloud. State-space models are capable of modeling large and complicated systems. However, it is impractical to use a single state space model to model the whole system due to the state space explosion problem.

Hierarchical models combine the combinatorial models and state-space models to evaluate the availability of large-scale systems. Wei et al. [24] constructs a hybird dependability model based on RBD and GSPN to model the virtual data center of cloud computing. Dantas et al. [25] combined RBDs and Markov models to analyze availability of Eucalyptus architecture. Because hierarchical model is decomposable, it solves the problem of state space explosion. However, the decomposability is not always manually controllable due to the automaticity of the model generation.

Besides the model based methods, there are a lot of measurement based methods to evaluate the availability. Fujita et al. [26] developed DS-Bench toolset to evaluate the dependability of a cluster of physical machines and a cloud computing environment. Sangroya et al. [27, 28] proposed a MapReduce benchmark suite to estimate the dependability and performance of MapReduce systems. Furthermore, there are also a number of prior works review the performance benchmarking for IaaS cloud [29, 30]. However, standard benchmarking solutions cannot be used directly for the prediction of cloud availability.

B. Tail Latency Evaluation

Tail latency, or response time, is another important metric reflecting the quality of user experience for online cloud services. Most existing work focus on evaluating the mean response time instead of the response time CDF [31, 32]. However, mean response time is far from sufficient to describe the user experience.

Generally, the response time consists of queuing time and service time. To evaluate waiting time, Sakuma et al. [33], construct a M/M/s queue model to analyze the tail approximation of the waiting time distribution of both patient and impatient customers. Bruneo et al. [13] also propose a tagged customer model based on SRN to calculate the waiting time distribution. The waiting time CDF is calculated by the probability that a tagged customer's request is absorbed and the probability of its corresponding initial state. However, the two above methods only calculate the waiting time CDF instead of the total response time CDF.

Muppala et al. [15] propose a tagged customer methods based on SRN to evaluate the response time CDF. Their model can apply to the closed queuing system with a fixed number of customers. However, most online cloud services are built on an open architecture and it can response to an arbitrary number of customer requests at the actual arrival rate. Grottke et al. [34] analyze the response time CDF using an open queuing network model. However, it is not easy to construct and solve the model due to the state space explosion problem.

VI. CONCLUSION

In this paper, we design an effective tagged customer model based on SRN to study the tail latency performance of online cloud services in the presence of failure-repair of the resources. In our method, the tagged customer model was used to analyze the CDF of online cloud service and predict the tail latency at any percentile. By solving the tagged customer model, we also can calculate the EA of online cloud services. We

conduct experiments by changing environment settings and compare model results to real experimental results to verify the accuracy of the proposed model. Experimental results show that the model results generally overestimates the response time at lower percentiles and underestimates the response time at higher percentiles. We also identified the potential factors that may impact the accuracy of online cloud services. It was found that by turning off the cache, increasing the search space, turning off Turbo Boost and configuring the DVFS performance governor, the prediction error can be reduced to as low as 2.45%. Taking these potential factor into account and modify our model accordingly may further reduce the prediction error, this is left as a future study.

In order to simulate the real utilization of the system whose request rate may vary significantly over time, we will further improve our model by adopting a Markov Modulated Poisson Process (MMPP) in the future.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000205, the National Natural Science Foundation of China under Grant No. 61402325. We would like to thank Binlei Cai, Zhuoxiao Zhang for their contributions. We would also like to thank Zitong Ji for the discussions on the initial drafts of this paper.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[3] I. Neamtiu and T. Dumitraş, "Cloud software upgrades: Challenges and opportunities," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, 2011, pp. 1–10.

[4] D. Krushevskaja and M. Sandler, "Understanding latency variations of black box services," in *22nd International World Wide Web Conference*, 2013, pp. 703–714.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[6] J. Nielsen, "The usability engineering life cycle," *IEEE Computer*, vol. 25, no. 3, pp. 12–22, 1992.

[7] L. Zhao and X. Zhou, "Slow or down?: Seem to be the same for cloud users," in *The first Workshop on Emerging Technologies for software-defined and reconfigurable hardware-accelerated Cloud Datacenters*, 2017, pp. 1–2.

[8] B. Cai, R. Zhang, X. Zhou, L. Zhao, and K. Li, "Experience availability: Tail-latency oriented availability in software-defined cloud computing," *Journal of Computer Science and Technology*, vol. 32, no. 2, pp. 250–257, 2017.

[9] Q. Lu, X. Xu, L. Zhu, L. Bass, Z. Li, S. Sakr, P. L. Bannerman, and A. Liu, "Incorporating uncertainty into in-cloud application deployment decisions for availability," in *IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 454–461.

[10] X. Xu, Q. Lu, L. Zhu, Z. Li, S. Sakr, H. Wada, and I. Weber, "Availability analysis for deployment of in-cloud applications," in *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*, 2013, pp. 11–16.

[11] R. Entezari-Maleki, K. S. Trivedi, and A. Movaghar, "Performability evaluation of grid environments using stochastic reward nets," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 204–216, 2015.

[12] T. Grainger, T. Potter, and Y. Seeley, *Solr in action*. Manning Cherry Hill, 2014.

[13] D. Bruneo, "A stochastic model to investigate data center performance and qos in iaas cloud computing systems," *IEEE Transactions on Parallel Distributed Systems*, vol. 25, no. 3, pp. 560–569, 2014.

[14] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, no. 2, pp. 82–98, 2014.

[15] J. K. Muppala, K. S. Trivedi, V. Mainkar, and V. G. Kulkarni, "Numerical computation of response time distributions using stochastic reward nets," *Annals of Operations Research*, vol. 48, no. 2, pp. 155–184, 1994.

[16] G. Ciardo, J. K. Muppala, and K. S. Trivedi, "SPNP: stochastic petri net package," in *International Workshop on Petri Nets and PERFORMANCE MODELS*, 1989, pp. 142–151.

[17] H. V. Karambelkar, *Scaling big data with Hadoop and Solr; 2nd ed.* Birmingham: Packt Publ., 2015.

[18] "Apache jmeter," http://jmeter.apache.org/, 2017.

[19] B. Wei, C. Lin, and X. Kong, "Dependability modeling and analysis for the virtual data center of cloud computing," in *Proc. High Performance Computing and Communications (HPCC)*, 2011, pp. 784–789.

[20] A. Volkanovski, M. Čepin, and B. Mavko, "Application of the fault tree analysis for assessment of power system reliability," *Reliability Engineering & System Safety*, vol. 94, no. 6, pp. 1116–1127, 2009.

[21] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, 2008.

[22] Q. Wu, M. Zhang, R. Zheng, Y. Lou, and W. Wei, "A qos-satisfied prediction model for cloud-service composition based on a hidden markov model," *Mathematical Problems in Engineering*, vol. 2013, no. 3, pp. 275–289, 2013.

[23] F. Longo, R. Ghosh, V. K. Naik, and K. S. Trivedi, "A scalable availability model for infrastructure-as-a-service cloud," in *Proc. the 41st IEEE/IFIP International Conference on Dependable Systems & Networks*, 2011, pp. 335–346.

[24] B. Wei, C. Lin, and X. Kong, "Dependability modeling and analysis for the virtual data center of cloud computing," in *Proc. High Performance Computing and Communications*, 2011, pp. 784–789.

[25] J. Dantas, R. Matos, J. Araujo, and P. Maciel, "Models for dependability analysis of cloud computing architectures for eucalyptus platform," *International Transactions on Systems Science and Applications*, vol. 8, pp. 13–25, 2012.

[26] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa, "Ds-bench toolset: Tools for dependability benchmarking with simulation and assurance," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012, pp. 1–8.

[27] A. Sangroya, D. Serrano, and S. Bouchenak, "Benchmarking dependability of mapreduce systems," in *Proc. the 31st IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 21–30.

[28] A. Sangroya, S. Bouchenak, and D. Serrano, "Experience with benchmarking dependability and performance of mapreduce systems," *Performance Evaluation*, vol. 101, pp. 1–19, 2016.

[29] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the source of tail latency through precise load testing and statistical inference." *Acm Sigarch Computer Architecture News*, vol. 44, no. 3, pp. 456–468, 2016.

[30] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[31] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An analytical model for multi-tier internet services and its applications," in *ACM SIGMETRICS Performance Evaluation Review*, 2005, pp. 291–302.

[32] N. Roy, A. S. Gokhale, and L. W. Dowdy, "Impediments to analytical modeling of multi-tiered web applications," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 441–443.

[33] Y. Sakuma, A. Inoie, K. Kawanishi, and M. Miyazawa, "Tail asymptotics for waiting time distribution of an m/m/s queue with general impatient time," *Journal of Industrial & Management Optimization*, vol. 7, no. 3, pp. 593–606, 2013.

[34] M. Grottke, V. Apte, K. Trivedi, and S. Woolet, "Response time distributions in networks of queues," *Queueing Networks*, vol. 154, pp. 587–641, 2011.

Tuning Multipath TCP for Interactive Applications on Smartphones

Quentin De Coninck*, Olivier Bonaventure
Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM)
Université catholique de Louvain
Louvain-la-Neuve, Belgium
Email: {quentin.deconinck,olivier.bonaventure}@uclouvain.be

***Abstract*—Multipath TCP enables smartphones to simultaneously use both WiFi and LTE to exchange data over a single connection. This provides bandwidth aggregation and more importantly reduces the handover delay when switching from one network to another. This is very important for delay sensitive applications such as the growing voice activated apps. On smartphones, user experience is always a compromise between network performance and energy consumption. However, the Multipath TCP implementation in the Linux kernel was mainly tuned for bandwidth aggregation and often wakes up the cellular interface by creating a path without sending data on it.**

In this paper, we propose, implement and evaluate MultiMob, a solution providing fast handover with low cellular usage for interactive applications. MultiMob relies on three principles. First, it delays the utilization of the LTE network. Second, it allows the mobile to inform the server of its currently preferred wireless network. Third, MultiMob extends the Multipath TCP handshake to enable immediate retransmissions to speedup handover. We implement MultiMob on Android 6 smartphones and evaluate its benefits by using both microbenchmarks and in the field measurements. Our results show that MultiMob provides similar latency as the standard Linux implementation while significantly lowering the cellular usage.

I. Introduction

Mobile devices such as smartphones are now an integral part of our digital life. Mobile data traffic continues to grow [1]. The performance of the WiFi and cellular networks have significantly increased over the last years. Compared with 3G, LTE provides both higher bandwidth and lower latency while WiFi reaches Gbps and more. These high bandwidth and low-latency networks encouraged the deployment of new applications. Mobile video benefited a lot from the bandwidth improvements. On the other hand, the lower latency enabled a new family of voice activated applications [21]. The user uses his/her voice instead of buttons to interact with the application that sends voice samples to the cloud. For such applications, latency is key and many protocols have been tuned during the last years to reduce it [4], [33].

For most smartphone users, the WiFi and cellular networks are not equivalent. WiFi has two major advantages compared to cellular networks. First, using WiFi consumes less energy [20], [26]. Second, most service providers charge for cellular data while most WiFi networks are free or charged on a flat-rate basis. For these reasons, many smartphones owners only use their cellular interface for voice calls and when there is no WiFi network available [7].

ISBN 978-3-903176-08-9 * FNRS Research Fellow

Multipath TCP is a recent TCP extension [16] that was designed with these smartphones in mind. Pluntke et al. [32] and then Raiciu et al. [35] first discussed the expected benefits of Multipath TCP on such mobiles devices. Later, Paasch et al. implemented handover features [30] in the Linux kernel [29]. Lim et al. showed the importance of taking energy consumption into account [23].

Industry has already adopted Multipath TCP on smartphones with two major deployments [3]. Apple uses Multipath TCP for its Siri voice activated application since 2013 and enables Multipath TCP for any application on iOS11 [7]. This is the largest deployment of Multipath TCP today [3] with about 700 million devices. There is another major deployment in Korea. In this country, high-end Android smartphones use Multipath TCP through network-operated SOCKS proxies to achieve Gbps [37].

Many authors studied and tuned the bandwidth aggregation capabilities of Multipath TCP [36], [22], [18], [27], [11], [6] in mobile networks. Although popular in the scientific literature, this is not the main use case for Multipath TCP on mobiles [7]. Smartphones rarely exchange large files [9] that would benefit from bandwidth aggregation. Measurement studies [13], [9] show that smartphones mainly use either short or long-lived intermittent TCP connections. Apple recently opened Multipath TCP on iOS11 mainly to provide seamless handovers and support interactive applications [7].

We first describe the current state-of-the-art of Multipath TCP on smartphones in Section II. We then propose MultiMob, a series of improvements that adapt Multipath TCP to the requirements of today's smartphone applications. More precisely, MultiMob provides a good compromise between latency and cellular usage.

A MultiMob server replies on the subflow used by the smartphone (§ III-A). If a smartphone sends a request over a cellular subflow because its WiFi subflow performs badly, the server should send its reply over the same subflow.

MultiMob minimizes cellular usage and unused subflows (§ III-B). Like iOS11 [7], MultiMob prefers to use the WiFi interface over the cellular one. MultiMob replaces the *make-before-break* strategy of the Multipath TCP implementation on Linux by *break-before-make*. With this strategy, the cellular

interface is only used after a failure of the WiFi one.

MultiMob limits handover delays (§ III-C). The *break-before-make* strategy minimizes energy consumption but at the expense of increased handover delays. MultiMob reduces those delays by extending the Multipath TCP protocol to carry data during the subflow handshake.

In Sect. IV, we collect measurements in a Mininet environment to assess MultiMob characteristics. In Sect. V, we evaluate MultiMob with real smartphones. Finally, Sect. VI concludes this paper. An extended technical report of this work is available [10].

II. State of the Art and Motivation

Multipath TCP [16] was designed with multihomed devices such as smartphones in mind. It enables them to exchange data belonging to a single connection over different network paths. It is described in details in [16], [36]. We briefly summarize its main features here. A Multipath TCP connection is in fact a combination of different TCP connections, called *subflows* in [16], that are grouped together. A Multipath TCP connection is established by using a three-way handshake as a regular TCP connection, except that the SYN packet contains the `MP_CAPABLE` option. This option negotiates the utilization of Multipath TCP and allows the client and server to exchange keys. Each Multipath TCP connection is identified by a token that is derived from the keys exchanged during the initial handshake [16]. Data can be exchanged over the initial subflow and both the client and the server can create additional subflows to use other paths or perform handovers. Those additional subflows must be established by using a four-way handshake with SYN packets that contain the `MP_JOIN` option. This option includes a token that identifies the corresponding Multipath TCP connection. Data can be transmitted over any of the available subflows. Multipath TCP uses two levels of sequence numbers. The standard TCP sequence and acknowledgement numbers are used in the TCP header to handle data sequencing and retransmissions on a per-subflow basis. Furthermore, Multipath TCP uses the Data Sequence Number (DSN) that tracks the position of the data in the bytestream. The DSN is placed inside the Data Sequence Signal TCP Option that also carries DSN acknowledgements. Thanks to this DSN, Multipath TCP can transmit data over one subflow and later retransmit it over another subflow because the initial one failed or became unresponsive. *Reinjecting* data over a different subflow is key to support handovers [36], [30].

There are two main implementations of Multipath TCP on mobile nodes: Apple's implementation on iOS [7] and the open-source Linux implementation [29]. We focus on the latter because it fully implements the protocol and can be easily modified. Besides supporting all the features described in [16], it includes several heuristics that are important for performance [36] without impacting interoperability.

A first component of the Multipath TCP implementation in the Linux kernel is the *path manager*. It determines when additional subflows must be created. The initial subflow is always established on the interface that points to the current default route. On a client, the default `fullmesh` path manager [29] creates new subflows immediately after the creation of the initial one and each time a new IP address is assigned to the client or learned from the server. This path manager does not initiate subflows from the server because it expects that the client's firewall will block incoming TCP connections.

A second component is the *packet scheduler*. It decides on which established subflow the next packet will be sent. The `default` scheduler extracts the smoothed round-trip-time of all the subflows whose congestion window is not full and selects the one having the lowest smoothed round-trip-time (RTT). Other schedulers more adapted to heterogeneous paths have been proposed [25], [28], [15].

Multipath TCP [16] also supports *backup subflows*. When a subflow is established, it is possible to set a bit in the `MP_JOIN` option to indicate that this subflow should not be selected by the scheduler to exchange data unless all non-backup subflows have failed. We observed that Siri in iOS11 [7] also sets the backup bit on the cellular subflow to discourage the utilization of the cellular interface to transport data. In the Linux Multipath TCP implementation, a subflow is considered to be in a *potentially failed* state once its retransmission timer expires. This subflow transitions to the active state as soon as new data is acknowledged on the subflow. The default scheduler uses a backup subflow if all the regular subflows are in the *potentially failed* state.

A. Multipath TCP on Smartphones

Before tuning Multipath TCP on smartphones, it is important to understand how they interact with the network. We summarize in this section some of the lessons we learned based on discussions with network operators and previous works.

Smartphone applications rarely perform bulk transfers Multipath TCP was designed to aggregate bandwidth and many articles evaluated whether Multipath TCP reaches that objective [31], [6], [11], [34]. However, smartphones rarely exchange very long files [14], [9]. Most of the connections carry a few KB. Many connections also experience large idle times [8]. While not being an issue from TCP perspective, from an energy viewpoint, it can consume energy if the radio needs to remain active to support it.

Many subflows do not carry data The `fullmesh` path manager immediately creates subflows on all active interfaces. However, most of these subflows are useless, i.e., *no* data is sent over them [9]. With the `default` scheduler, if the initial subflow exhibits a lower RTT than the additional ones, Multipath TCP will only use the initial one. Previous works also indicate than Multipath TCP can perform worse than TCP on short flows in heterogeneous networks [18], [27].

Mismatch with user expectations Most users favor WiFi over cellular for both monetary and power consumption reasons [20], [7]. They expect that their smartphone will use WiFi whenever it works well and will switch to cellular only if it brings some benefits. However, the packet scheduling decision is taken by the sender of the packet. In practice, smartphones mainly receive data [14], [9], meaning that most of the

scheduling decisions are taken by remote servers. Because the measured round-trip-time is the only metric, the server scheduling decision can go against the user expectations.

Backup subflows consume energy One way to minimize the utilization of the cellular network is to always establish the cellular subflow as a backup subflow [16], [23]. While useful in mobility scenarios, there is no point to create backup subflows if the primary one does not face any connectivity issue. Indeed, energy consumption is a major concern for mobile devices [7], [5], [2]. However, opening a subflow on the cellular interface without using it is expensive from an energy consumption viewpoint [11], the WiFi interface consuming at least five times less than the LTE one [26]. In the remaining of this paper, we use the LTE model proposed by Huang et al. [20] to estimate the cellular power consumption (we expect similar results with other models [26]). In the model used [20], opening one cellular subflow on a smartphone is equivalent to lighting up the screen 100% during the `RRC_CONNECTED` period, which lasts around 11 s. Opening preventively the cellular subflow as proposed in [30] is thus very expensive from an energy consumption viewpoint. Siri in iOS11 still creates cellular subflows at the beginning of the connection.

Related Works Lim et al. [24] proposed eMPTCP that delays the use of the cellular below a given threshold of bytes transfered and opens the cellular subflow if the WiFi bandwidth is not sufficient. While working with bulk transfers, interactive applications can transmit very few bytes and do not need large bandwidths. Sinky et al. [38] proposes to rely on the signal strength of the WiFi network to tune the congestion window to trigger seamless WiFi handover with bulk transfer. However, it was only tested under NS3-DCE environment and not with actual devices. Han et al. [17] proposes to disable the cellular when the WiFi is sufficient with delay-tolerant traffic. However, interactive traffic is delay-sensitive.

III. Tuning Multipath TCP

We explain how MultiMob improves Linux Multipath TCP. We first add to the server's packet scheduler a heuristic that enables it to infer the wireless conditions that affect the client subflows. Second, we implement an oracle that monitors the network and opens cellular subflows only when needed. Third, we extend the Multipath TCP protocol so that a client can retransmit data inside the SYN that is used to create an additional subflow during a handover.

A. Towards Global Scheduling

When a Multipath TCP connection is composed of 2 or more subflows, each of the communicating hosts independently selects the best subflow to transmit each data. The Linux implementation selects the available subflow with the lowest round-trip-time (RTT). This scheduler works well in a variety of environments [31]. However, selecting subflows only on the basis of their RTT is not always the best solution. Consider a smartphone user that moves while using the Siri application. This application regularly sends small bursts of data and the server returns responses. If the smartphone detects that the WiFi starts to be lossy, it will start to send data over the cellular subflow. However, the server is not aware of the movement of the smartphone and its packet scheduler still sends responses over the WiFi subflow because it has the lowest RTT. The server will only switch to the cellular subflow after the expiration of its retransmission timer, which potentially wastes hundreds of milliseconds.

To solve this problem, MultiMob includes a packet scheduler that uses the most recent data sent by the smartphone as a hint to select the most suitable subflow. On the smartphone, MultiMob uses a priority scheduler that favors WiFi and only uses cellular when the WiFi subflow experiences retransmissions. The server-side scheduler maintains for each subflow the timestamp of the *last original packet* received over this subflow. A packet is considered to be *original* if it contains new data (based on its DSN) or if it successfully concludes a subflow establishment. Similarly, an acknowledgement is considered to be original if its Data ACK advances the lower edge of the sending window. The `MultiMob` scheduler first removes from consideration the potentially failed subflows and the ones where this data has already been transmitted. Then it iterates over all remaining subflows to identify the one having the most recent original reception. If the congestion window of this subflow is not full, it is selected.

Thanks to this scheduler, the server can quickly detect the most suitable subflow while taking into account subflow backup preferences. For an interactive application like Siri that sends small requests, the server will always reply on the subflow that was last used by the client.

B. Break-Before-Make

In the Linux kernel implementation, when a Multipath TCP connection starts, the `fullmesh` path manager opens the connection over the primary interface and then creates subflows over the other ones. If the cellular interface is configured as a backup interface, data packets will only be sent over this interface once the WiFi interface fails. This *make-before-break* approach minimizes the amount of data sent over the cellular interface. Unfortunately, it does not minimize the energy consumption. There is no significant difference from an energy consumption viewpoint between a cellular interface that transmits only SYN/FIN or several data packets.

MultiMob opts for *break-before-make* and creates subflows over the backup interface after having detected failures on the primary interface. With *break-before-make*, the key issue from a performance viewpoint becomes how quickly can the smartphone detect that a wireless interface works badly and new backup subflows must be created. MultiMob detects those failures through a *Multipath TCP oracle* implemented as a kernel module. The oracle relies on the assumption that if a network interface experiences connectivity issues, subflows using it will experience retransmissions and losses, even if they belong to different connections. To track those events, our oracle maintains a monitoring table of netpaths. A *netpath* is a tuple (IP_{src}, IP_{dst}, network interface). We aggregate the information on a per layer-3 flow basis to reduce the size of the

monitoring table. This structure is well adapted to deployments with SOCKS proxies such as [37] where all Multipath TCP connections are terminated on the proxy.

Our oracle computes every T_s seconds statistics based on the subflows associated to a given netpath. Our current implementation collects three metrics: smoothed loss rate ($sloss$), smoothed retransmissions rate ($sretrans$) and maximum RTO. Those statistics are computed based on the per-subflow state maintained by the kernel. It also takes into account Tail Loss Probes [12]. When the TLP timer fires, we enter FACK mode and the packet at the head of the write queue is marked as lost. The smoothed rates are computed by using Volume-weighted Exponential Moving Averages (V-EMA) used by Android to estimate the loss rate of WiFi beacons. These V-EMA reduce to the three following equations

$$val_{i+1} = \frac{prod_{i+1}}{vol_{i+1}} \quad (1)$$

$$prod_{i+1} = \alpha(val_{new} \cdot vol_{new}) + (1-\alpha)prod_i \quad (2)$$

$$vol_{i+1} = \alpha \cdot vol_{new} + (1-\alpha)vol_i \quad (3)$$

where val_{new} is the new value of the studied metric (e.g., lost sent packets during the last T_s period), vol_{new} is the volume of this new value (e.g., total number of packets sent during last T_s period), $prod_i$ is the product at iteration i, vol_i the volume at iteration i and val_i the value at iteration i. $prod_0 = vol_0 = val_0 = 0$ and no value is computed if vol_{new} is 0. $\alpha \in [0, 1]$ is a parameter experimentally set to 0.5 as in Android.

The MultiMob oracle sets thresholds to detect underperforming netpaths. Once a threshold is crossed, MultiMob triggers the creation of backup subflows for all connections associated to the underperforming netpath. It also marks the subflows associated with that netpath as potentially failed. Since the oracle is part of the kernel, it can query the state of all established Multipath TCP connections and trigger backup subflows creation once a problem is detected.

The last scenario that we consider is a client-initiated download. During such a download, the server pushes data towards the client. If a subflow fails, the client stops receiving data, but it is difficult for the Multipath TCP stack to distinguish between losses in the network and the server application becoming idle for any reason. We modify Multipath TCP so that the server can indicate to the client that a data transfer is not yet finished. This can work with existing applications. We define two signals. The first one is sent in the Multipath TCP DSN option. We modify one of the unused bits of the DSN option that we call the `MP_IDLE` bit. This bit is set by server when it sends a data packet that empties its send buffer. Otherwise, the `MP_IDLE` bit of the DSN option is reset. Since this bit is included in the DSS option, it is sent reliably to the client. A receiver should not expect a connection to be idle unless it has received a DSN option with the `MP_IDLE` bit set. We also define a new experimental Multipath TCP option that carries the current value of the RTO. The client sends an empty RTO option from time to time and the server returns the same option containing its current retransmission timer. The client uses this information to set its idle timer at $max(500 \text{ msec}, RTO_{Server})$. This timer runs while the `MP_IDLE` flag of the last received data is reset. It is reset every time a packet is received on the subflow and stopped if the last data packet had the `MP_IDLE` flag set. If the timer expires, the oracle triggers the creation of backup subflows.

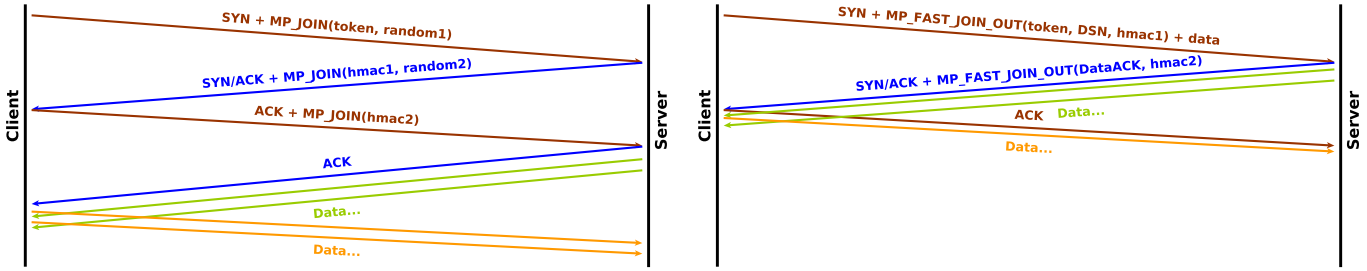
C. Immediate reinjections

The *break-before-make* approach described in the previous section is beneficial from an energy viewpoint. However, a mobile typically detects the failure of a wireless interface by the expiration of its retransmission timer or TLP probe. This unacknowledged data can only be retransmitted over another interface once a subflow has been established over this interface. Multipath TCP [16] requires a four-way handshake before allowing the transmission of data from the server. This handshake has two purposes. First, it creates state on the endpoints (and possibly on the intermediate middleboxes). Second, the client and the server authenticate each other. This authentication is performed by using the keys exchanged during the initial handshake. Both the client and the server exchange HMACs computed over these keys and random numbers exchanged in the SYN and SYN+ACK (see Fig. 1a). Unfortunately, this handshake delays the reinjection of the lost data since the client cannot send data before having received the fourth ACK [16].

To reduce this delay, we modify Multipath TCP to support the transmission of data inside SYN or SYN+ACK packets. For this, we define two new Multipath TCP options: `FAST_JOIN_IN` (`FJI`) and `FAST_JOIN_OUT` (`FJO`). This is different than TCP Fast Open [33] because a new subflow is established between hosts that already share state for one Multipath TCP connection.

A naive approach would be to simply place data inside the SYN and require the server to accept this data immediately. Unfortunately, this solution would cause security problems because this SYN is not authenticated. The `MP_JOIN` option that it carries contains only the 32 bits token that identifies the connection and a random number that is used to authenticate the server (Fig. 1a). This token is not sufficient to authenticate the client because a passive listener could have observed it during the establishment of a previous subflow for the same connection, e.g., on an open WiFi network.

The `FJO` option described on Fig. 1b solves this problem and allows the client to carry authenticated data in the initial SYN. Our `FJO` option contains three fields. The token identifies the Multipath TCP connection as in the `MP_JOIN` option. The Data Sequence Number (DSN) indicates the sequence number of the data contained in the SYN payload. The third field is a HMAC computed over the connection keys exchanged during the initial handshake and the DSN. This last field ensures that the initiator of the subflow is one of the connection hosts. To prevent replay attacks, our implementation only accepts one SYN containing the `FJO` option for a given DSN. To cope with lost acknowledgments, if $EDSN$ is the next expected DSN on the server and $SDSN$



(a) Multipath TCP uses a four-way handshake that lasts two round-trip-times to create an additional subflows with `JOIN`.

(b) With `FAST_JOIN`, the client can immediately send data inside the SYN packet.

Fig. 1: Time-sequence diagrams for the establishment of additional subflows.

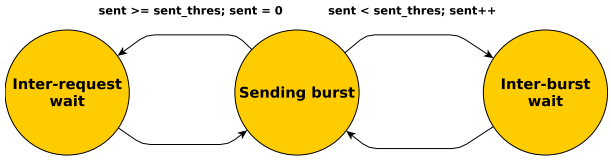


Fig. 2: State machine of a simple interactive application.

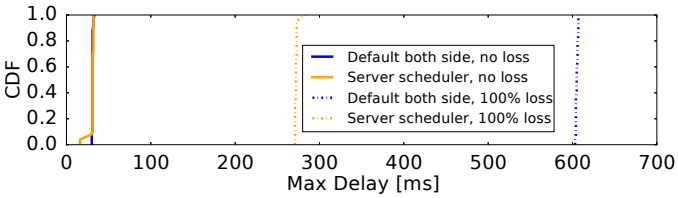


Fig. 3: Maximum delay to return an answer to simplified interactive requests. Each scenario ran 25 times. Losses begin to occur when the client is in *inter-user interaction wait* state, i.e., between request bursts.

the DSN contained in the `FAST_JOIN` SYN, the server allows $SDSN$ to be in the range $[EDSN - rcv\_wnd, EDSN]$ where rcv_wnd is its receive window. Once the SYN has been acknowledged, the server can immediately start to send data to the client.

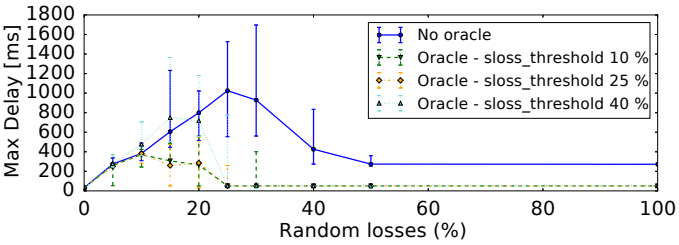
The `FJO` option is useful when the mobile client sends data to a server. However, there are situations where the server pushes data towards the client. A typical example are streaming applications where the server pushes data at a regular rate. When the oracle running on the mobile client detects losses or the absence of data, it may want to quickly establish a subflow without having data to send to the server. This case is covered with the `FJI` option [10] (not shown for space limitations). This option is very similar to the `FJO` option, except that it contains the current Data ACK instead of a DSN, with the HMAC computed over this Data ACK. With this new option, the server can authenticate the client immediately and send data upon reception of the SYN. By using the `FJI` option, the data transfer can resume after 1 RTT, instead of 2 RTTs with normal join.

IV. Emulations

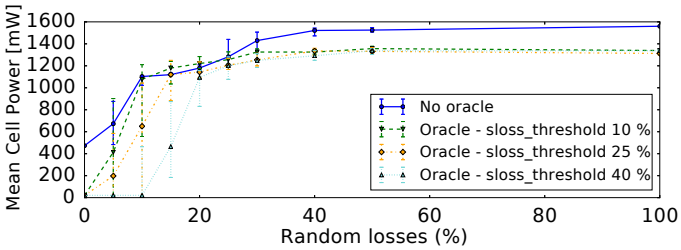
We evaluate in this section the performance of MultiMob in Mininet environments [19] in a scenario with two disjoint paths between the client and the server. Emulations are based on Multipath TCP v0.91 in Linux 4.1.

a) Studied Traffic: Siri is a famous example of interactive traffic. However, it is not open-source and only runs on iOS devices. To evaluate the interactions between a simple interactive application and Multipath TCP, we use a simplified model based on an analysis of the behavior of Siri. Our model is a three-states process shown in Fig. 2. The client maintains a counter: `sent`. In the *sending burst* state, the client sends a burst of 2500 bytes using packets carrying between 50 and 500 bytes. Then `sent` is incremented and the client waits in the *inter-burst wait* state before going back to *sending burst*. Once `sent` reaches `sent_thresh`, the client switches to the *inter-user interaction wait* state that represents the random delay between successive user interactions. `sent_thres`, *inter-burst wait* and *inter-user interaction wait* are empirically set to 9, 1/3 s and 5 s respectively. We model the server as a process that returns a 750 bytes response after each burst. Our simple client application then collects the delay between each request and the server's response. Unless stated, all the measurements in this paper are based on this traffic.

b) MultiMob Scheduler: The primary path exhibits a RTT of 15 ms and the additional one 25 ms. Both paths have a bandwidth of 10 Mbps and the router queue sizes are equal to the bandwidth-delay product. The client opens the connection over the primary path and then creates a backup subflow over the additional one. Figure 3 shows that when there are no losses, the `default` and `MultiMob` schedulers running on the server exhibit quite similar performances. Notice that because of the default `tcp_slow_start_after_idle` set to 1, a request can be answered in two RTTs if its sending phase generates more packets than the initial congestion window (10 packets). However, when the primary subflow fails between two requests, the `MultiMob` scheduler reduces the maximal delay experienced by a factor of two. When the client sends its first request after a loss, it experiences a RTO before reinjecting the packet on the additional subflow. However, with the `default` scheduler, the server does not



(a) Maximal delay to answer a simplified interactive request.



(b) Estimated mean cell power consumption based on model [20].

Fig. 4: Simplified interactive requests with light continuous background traffic. The second interface is set as backup. If any, the loss event occurs while the client is in *inter-user interaction wait* state, i.e., between request bursts. Markers shows medians and error bars 25th and 75th percentiles over 25 runs.

know that the primary subflow failed, and it sends the reply to the primary lossy subflow and experiences a RTO too before reinjecting on the additional subflow. Since the `MultiMob` server-side scheduler follows the last client decision, it does not experience the RTO at the server side.

c) Influence of Threshold Value: To assess the benefits of the oracle and determine the threshold value for $sloss$, we rely on simplified interactive requests while a light background request/response traffic (12 KB/s) is present. Figure 4a shows the maximal latency to answer a request and Fig. 4b shows the estimated mean power consumed on the second path. The energy consumption is estimated by using the packet trace and the model presented in [20], considering that the cellular interface is always powered on. Without losses, we observe similar requests delays, while the backup subflow is not established with the oracle. When losses occur on the primary path, the oracle knows that the background traffic experiences connectivity issues and creates backup subflows for all connections using that path. Then, the simulated interactive client can directly use the additional path and does not face RTO. Since the server uses `MultiMob` scheduler, it replies on the subflow used for the request and no RTO occurs. On the contrary, the interactive connection must face a RTO if there is no oracle before using the additional path, even if the additional subflow is always established at the beginning of the connection. Furthermore, when the link is very flappy (20-30% losses), the case without the oracle tries to reuse the lossy path once some ACKs manage to reach the host, while the oracle prevents this behavior. With the oracle the creation of the additional subflow depends on the network conditions and the $sloss$ threshold. When set to a low value, e.g., 10%, delays remain low but a few losses suffice to open the additional subflow. With higher values like 40%, the additional path remains closed in the median case when the primary path experiences 10% of random losses, but it can experience higher latencies. Based on those simulations, we experimentally set the $sloss$ threshold to 25% as a reasonable trade-off between low-latency and low additional path use. $sretrans$ is set to 50% and the max RTO threshold is empirically set to 1.5 seconds to avoid using a subflow that might hurt interactivity because of lack of retransmission reactiveness.

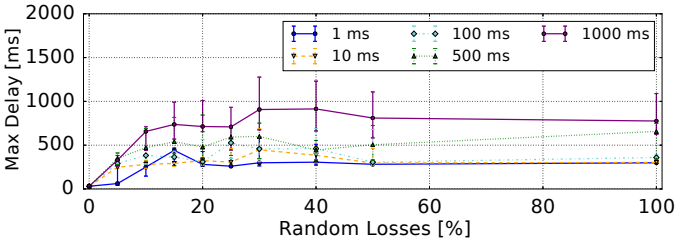


Fig. 5: Varying T_s for interactive traffic, with $sloss$ set to 25%. Showing 25th, 50th and 75th percentiles.

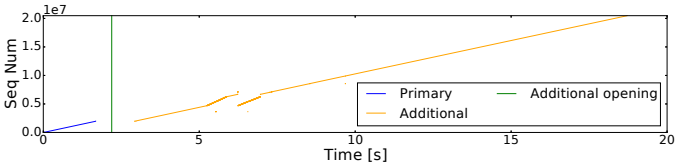


Fig. 6: Time-sequence graph of the packets received by a client during a 20 MB HTTP GET. The primary subflow suffers from 100% losses at 1.5 s. Retransmissions at 6 second are caused by a burst of duplicate ACKs.

d) Influence of the Oracle Periodicity: The reactivity of the oracle also depends on the oracle timer T_s. Indeed, as shown in Fig. 5, the lower T_s, the quicker the reaction of the oracle to losses and the lower the variability of the detection. A value of 1 ms allows very quick reaction, but the oracle might spend a lot of CPU time to update its monitoring table. In the remaining of the paper, T_s is empirically set to 500 ms to match sub-second reactivity and low CPU usage on mobiles.

e) Bulk Download and Primary Subflow Loss: The client downloads a 20 MB file and we add 100% losses on the primary path after 1.5 s. Fig. 6 shows that after some idle time, the client detects that it did not receive data and triggers the creation of a second subflow. The server then starts to use the new subflow and the data transfer continues.

f) Fast Join Benefits: We evaluate the benefits of using the `FAST_JOIN_OUT` option with regular request/response traffic over an emulated network where the primary path experiences 100% losses after five seconds. Figure 7 shows the difference of the delays of the first request following the

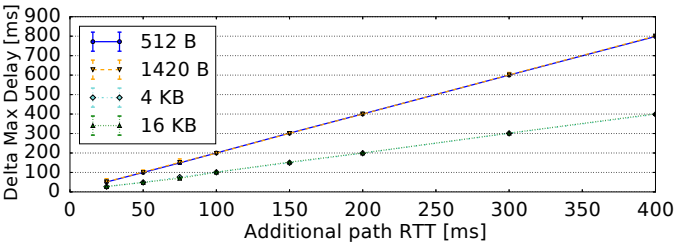


Fig. 7: Delta of max delay between normal and fast joins depending on the request size. Markers are medians over 6×2 runs, bars show min and max.

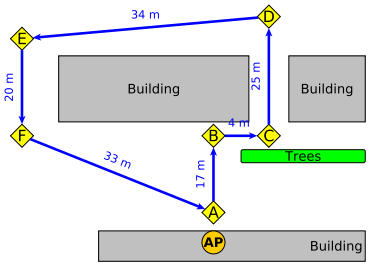


Fig. 8: Walk map for micro-benchmarks.

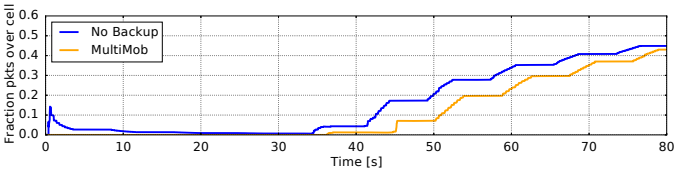
loss event using normal and fast joins. If the request fits inside one TCP packet, as for the popular Siri application, the fast joins provide immediate reinjections when the client sends data and the response can be received after one RTT.

V. Performance Evaluation

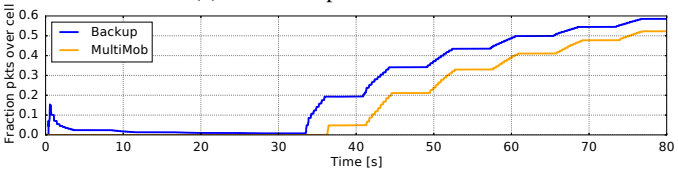
This section presents the evaluation of MultiMob on Nexus 5 smartphones running Android 6.0.1. For this, we backported Multipath TCP v0.89 to the Linux 3.4 kernel of Nexus 5 phones. Three configurations are studied: 1) *No backup* (*NBK*), MPTCP with the `fullmesh` path manager; 2) *Backup* (*BK*), NBK with backup subflows on cellular; and 3) *MultiMob*, the proposed solution described in Sect. III. We use two servers. The first one, configured with the `default` scheduler, is used by NBK and BK. The second one, configured with the `MultiMob` server-side scheduler, is used by MultiMob. In this section, we first explore particular use cases with micro-benchmarks to understand the benefits of MultiMob. We then compare at a larger scale NBK and BK with MultiMob through active measurements performed on a set of modified Android 6 smartphones used by real users.

A. Mobility Micro-Benchmarks

To evaluate how MultiMob performs in changing wireless conditions, we go for a short walk (Figure 8) with two smartphones. The first uses MultiMob and the other a vanilla Multipath TCP configuration. Our walk starts at **A**, close to the WiFi AP. Starting from **C**, the WiFi signal becomes weaker given the distance and the presence of trees and buildings. Android usually detects the loss of the WiFi signal and tears down the WiFi network at location **D**. Starting at location **F**, the WiFi signal becomes available again.



(a) No Backup vs. MultiMob.



(b) Backup vs. MultiMob.

Fig. 9: Evolution of the mean fraction of total packets carried by the cellular network for the simplified interactive traffic.

Configuration	MD (ms)	RA	CP (mW)
No backup	1112	100	884
Backup	780	100	885
MultiMob	1183	100	657

TABLE I: Aggregated results from simulated interactive micro-benchmarks. MultiMob shows the mean value over both runs. MD = Max Delay, RA = Requests Answered, CP = mean Cell Power consumption.

Simulated Interactive Traffic Our test phones send 100 requests during our 80 s walk from **A** to **D**. Figure 9 shows the instantaneous mean over the test duration of the fraction of total packets that are carried by the cellular interface for the two runs. In addition, Tab. I shows aggregated results related to these tests. With NBK and BK, the cellular subflow is always created at the beginning of the connection, but no data packet is sent on the cellular subflow while the WiFi signal remains good. This is expected for the backup case, and the larger RTT on the cellular network combined to the low network load explain the NBK results. When the client requests start to be lost between locations **C** and **D**, the cellular network is used to recover the connectivity. Since the cellular subflow was established at the beginning of the connection, the NBK and BK cases often experience a lower maximal delay than MultiMob. Since the cellular subflow is already established, the NBK and BK cases can reinject requests on the cellular subflow as soon as a RTO occurs on the WiFi subflow. MultiMob needs first to detect the connectivity loss with its oracle before establishing the cellular subflow, but its maximum delay remains similar to those of NBK and BK cases[1]. On the opposite, MultiMob consumes less cellular energy since it delays the utilization of the cellular interface.

Fixed Rate Streaming Traffic For this test, we configure the smartphones to stream a web radio over HTTP while performing twice the walk presented in Fig. 8. Our servers relay the same web radio at a fixed bitrate using Icecast.

[1]It is actually very dependent of the wireless conditions of a particular run. The lowest max delay observed for MultiMob over runs was 599 ms.

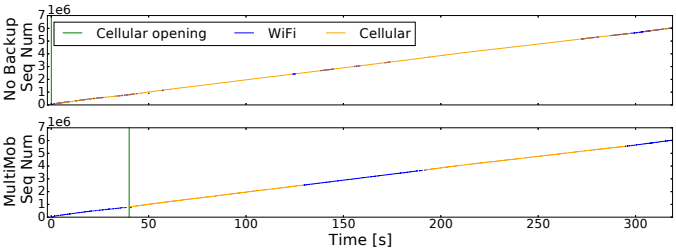


Fig. 10: Time-sequence graph of the server streaming flow as perceived by the client for the No Backup vs. MultiMob. Color indicates on which interface packet was received.

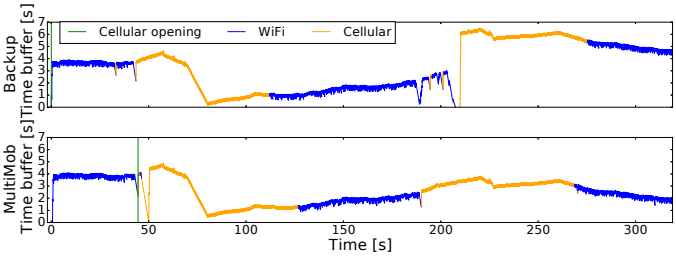


Fig. 11: Playing time of the client buffer for the worst case in Backup vs. MultiMob test. Color indicates on which interface packet was last received.

Since all the data flows from the server to the client, all the scheduling decisions are made by the server.

Figure 10 shows the time-sequence graph for the NBK vs. MultiMob test. We observed no stall during those experiments. However, the NBK case sends data nearly exclusively on the cellular interface, even when the WiFi network is available. From the server perspective, the cellular network appears to be more stable with an often lower estimated RTT than the WiFi one due to motion. This explains why the `default` scheduler prefers the cellular subflow. MultiMob forces the server to use the WiFi when it is still available. The WiFi to cellular (between **C** and **D**) and the cellular to WiFi (between **F** and **A**) handovers are visible on the MultiMob graph. Furthermore, notice that MultiMob waits 40 s before opening the cellular subflow using `FJI`, when the receive timer detects that no more data is received after some time without having received a `MP_IDLE`. Based on our model [20], NBK consumed 444 J for the cellular interface during the test (1386 mW), while MultiMob spent 329 J (1028 mW).

In the BK vs. MultiMob test, the network interface usage is similar, i.e., WiFi is used when available. Over a dozen of runs we observed no stall, except for a test that impacted both Backup and MultiMob. The buffer playing time at client side for that test is shown on Fig. 11. At 50 s (first **C**-**D** pass), MultiMob faces an half-second stall time, due to the reception of a packet on the WiFi network while the cellular subflow was already established. Since packets are acknowledged on the subflow they came from, the `MultiMob` server-side scheduler then tries to reply on the WiFi subflow, but it was meanwhile lost. After facing a RTO, the server reinjects this reply on the cellular subflow and the connection continues. The BK case experienced a 3 s stall time at time 205 s (second **C**-**D** pass). This stall was caused by the `default` scheduler that favors the WiFi subflow over the backup subflow on the cellular interface. Indeed, after 200 s, the WiFi was underperforming, the server experienced RTOs and reinjected data on the cellular subflow, but it then came back. When the WiFi signal eventually disappeared, the RTO value increased because of previous losses and the RTO expired seconds after the actual WiFi loss. Again, the BK case opened the cellular path at the beginning of the connection, while this happened at 45 s by MultiMob. Furthermore, MultiMob has a smaller cellular energy consumption with 319 J (994 mW), though the BK one remains close with 347 J (1083 mW).

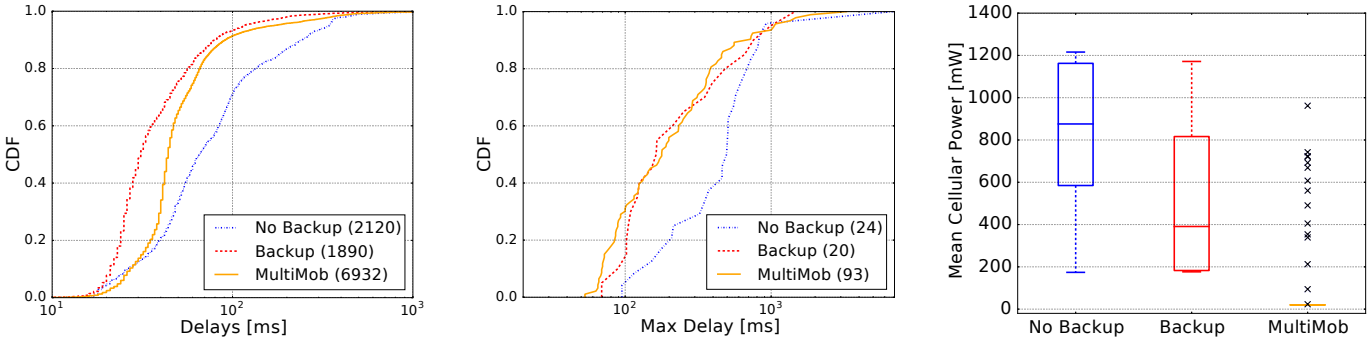
B. Measurements with Real Users

This section summarizes active measurements performed on Nexus 5 devices distributed to a few students and academics over a period of seven weeks (28th January - 22nd March 2017). We installed on each smartphone an Android application that periodically changes the network configuration, either once during night or after a reboot. Our measurement application runs in the background and sends data when it detects that the smartphone moves. Network conditions of tests depend on the presence of WiFi and/or LTE networks. To observe the performance of MultiMob to switch from the WiFi network to the cellular one, we only consider here tests where both WiFi and cellular interfaces are online at the beginning of the tests. Notice that WiFi can be lost during some tests.

Figure 12a shows that nearly all simplified interactive requests are answered within one second. Notice that simultaneously using two paths for such traffic as with the NBK can lead to increased response delays because of network heterogeneity between paths. Figure 12b plots the maximum delay observed during the tests. For all configurations, the maximum delay observed to answer simplified interactive requests remains within one second, with rare outliers higher than two seconds. Though the oracle detection to trigger cellular subflow is the largest delay component, MultiMob does not impact too much the request traffic when WiFi is lost. The main difference between MultiMob and both NBK and BK resides in scenarios where the WiFi remains alive during the whole test. The energy consumption computed on the entire traffic collected during the test is shown in Fig. 12c. Since NBK and BK always open additional subflows at the beginning of the connection, they consume energy, even if no real data is sent on that interface. Background connections initiated by real users can sometimes increase energy consumption by using the cellular interface. In contrast, since most of the time MultiMob does not create additional subflows, its cellular energy consumption is very low. MultiMob can thus keep low latency for delay-sensitive applications while limiting energy impact of Multipath TCP.

VI. Conclusion

Given that smartphones have both cellular and WiFi interfaces, users expect them to be able to perform seamless



(a) All request delays. The legend contains number of requests for the configuration.

(b) Maximal delays. The legend shows the number of tests for the configuration.

(c) Estimated mean cellular power [20] during the test when primary path stays alive.

Fig. 12: Simplified interactive traffic with real users (easier to see with color).

handovers between those two network interfaces. Multipath TCP enables such seamless handovers since it can use both cellular and WiFi interfaces for a single connection. Using both interfaces simultaneously is too expensive from an energy viewpoint. We propose, implement and evaluate MultiMob, a set of improvements to the Multipath TCP implementation and protocol. MultiMob uses *break-before-make* to minimise energy consumption. It extends Multipath TCP to support immediate retransmissions over a different interface. Furthermore, thanks to its scheduler, a server automatically selects the best performing interface to respond to requests from a smartphone. Our measurements indicate that MultiMob improves the performance of Multipath TCP on smartphones while minimizing energy consumption.

MultiMob is available: http://multipath-tcp.org/multimob

References

[1] Cisco visual networking index, February 2010.

[2] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *IMC'09*, pages 280–293. ACM, 2009.

[3] Olivier Bonaventure and SungHoon Seo. Multipath TCP deployments. In *IETF Journal*, volume 12, pages 24–27. November 2016.

[4] Bob Briscoe et al. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, 2014.

[5] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *USENIX'10*, volume 14. Boston, MA, 2010.

[6] Yung-Chih Chen et al. A measurement-based study of MultiPath TCP performance over wireless networks. In *IMC'13*, pages 455–468. ACM.

[7] Stuart Cheshire et al. Advances in networking, part 1. https://developer.apple.com/videos/play/wwdc2017/707//, June 2017.

[8] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. Observing real smartphone applications over Multipath TCP. *IEEE ComMag*, 54(3):88–93, March 2016.

[9] Quentin De Coninck, Matthieu Baerts, Benjamin Hesmans, and Olivier Bonaventure. A first analysis of Multipath TCP on smartphones. In *PAM'16*, pages 57–69. Springer, 2016.

[10] Quentin De Coninck and Olivier Bonaventure. Every millisecond counts: Tuning Multipath TCP for interactive applications on smartphones. Technical report. Available at http://hdl.handle.net/2078.1/185717.

[11] Shuo Deng et al. Wifi, lte, or both? measuring multi-homed wireless internet performance. In *IMC'14*, pages 181–194. ACM, 2014.

[12] N Dukkipati et al. Tail Loss Probe (TLP): An algorithm for fast recovery of tail losses. *IETF Draft, draft-dukkipati-tcpm-tcploss-probe-01*, 2013.

[13] Hossein Falaki et al. Diversity in smartphone usage. In *MobiSys'10*, pages 179–194. ACM, 2010.

[14] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *IMC '10*, pages 281–287, New York, NY, USA, 2010. ACM.

[15] Simone Ferlin et al. Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 431–439. IEEE, 2016.

[16] A. Ford et al. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[17] Bo Han et al. Mp-dash: Adaptive video streaming over preference-aware multipath. In *CoNEXT'16*, pages 129–143. ACM, 2016.

[18] Bo Han, Feng Qian, Shuai Hao, and Lusheng Ji. An anatomy of mobile web performance over Multipath TCP. In *CoNEXT '15*, pages 5:1–5:7, New York, NY, USA, 2015. ACM.

[19] Nikhil Handigol et al. Reproducible network experiments using container-based emulation. In *CoNEXT'12*, pages 253–264. ACM, 2012.

[20] Junxian Huang et al. A close examination of performance and power characteristics of 4g lte networks. In *MobiSys'12*, pages 225–238. ACM.

[21] Will Knight. Conversational interfaces. *MIT Technology Review*.

[22] Ming Li et al. Multipath transmission for the internet: A survey. *IEEE Communications Surveys Tutorials, vol. PP*, (99):1–41, 2016.

[23] Yeon-sup Lim et al. How green is Multipath TCP for mobile devices? In *All Things Cellular'14*, pages 3–8. ACM, 2014.

[24] Yeon-sup Lim et al. Design, implementation, and evaluation of energy-aware Multi-Path TCP. In *CoNEXT'15*, page 30. ACM, 2015.

[25] Yeon-sup Lim et al. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *CoNEXT'17*, pages 33–34. ACM, 2017.

[26] Ana Nika et al. Energy and performance of smartphone radio bundling in outdoor environments. In *WWW'15*, pages 809–819. ACM, 2015.

[27] Ashkan Nikravesh et al. An in-depth understanding of Multipath TCP on mobile devices: Measurement and system design. In *Mobicom'16*, pages 189–201. ACM, 2016.

[28] Bong-Hwan Oh and Jaiyong Lee. Constraint-based proactive scheduling for mptcp in wireless networks. *Computer Networks*, 91:548–563, 2015.

[29] Christoph Paasch, Sebastien Barre, et al. Multipath TCP in the linux kernel. http://www.multipath-tcp.org, 2017.

[30] Christoph Paasch et al. Exploring mobile/wifi handover with Multipath TCP. In *CellNet'12*, pages 31–36. ACM, 2012.

[31] Christoph Paasch et al. Experimental evaluation of Multipath TCP schedulers. In *CSWS'14*, pages 27–32. ACM, 2014.

[32] Christopher Pluntke et al. Saving mobile device energy with Multipath TCP. In *MobiArch'11*, pages 1–6. ACM, 2011.

[33] Sivasankar Radhakrishnan et al. TCP Fast Open. In *CoNEXT'11*, page 21. ACM, 2011.

[34] Costin Raiciu et al. Improving datacenter performance and robustness with multipath tcp. In *CCR'11*, volume 41, pages 266–277. ACM, 2011.

[35] Costin Raiciu et al. Opportunistic mobility with Multipath TCP. In *MobiArch'11*, pages 7–12. ACM, 2011.

[36] Costin Raiciu et al. How hard can it be? designing and implementing a deployable Multipath TCP. In *NSDI'12*, pages 29–29, 2012.

[37] SungHoon Seo. Kt's giga lte. IETF 93, 2015.

[38] Hassan Sinky et al. Proactive Multipath TCP for seamless handoff in heterogeneous wireless access networks. *IEEE Transactions on Wireless Communications*, 15(7):4754–4764, 2016.

Waypoint Routing in Special Networks

Saeed Akhoondian Amiri[1] Klaus-Tycho Foerster[2] Riko Jacob[3] Mahmoud Parham[2] Stefan Schmid[2]
[1] MPI Saarland, Germany [2] University of Vienna, Austria [3] IT University of Copenhagen, Denmark

***Abstract*—Waypoint routing is a novel communication model in which traffic is steered through one or multiple so-called waypoints along the route from source to destination. Waypoint routing is used to implement more complex policies or to compose novel network services such as service chains, and also finds applications in emerging segment routing networks. This paper initiates the study of algorithms and complexity of waypoint routing on special networks. Our main contribution is an encompassing characterization of networks on which routes through an arbitrary number of waypoints can be computed efficiently: We present an algorithm to compute waypoint routes for the important family of outerplanar networks, which have a treewidth of at most two. We show that it is difficult to go significantly beyond the graph families studied above, by deriving NP-hardness results on slightly more general graph families (namely graphs of treewidth three). For the case that the number of waypoints is constant, we also provide a polynomial-time algorithm for any constant treewidth network, even if waypoints change the flow sizes. For arbitrary numbers of waypoints however, the constraint of different flow-sizes between waypoints turns the problem hard, already if the network contains just a single cycle. Finally, we extend the study of waypoint routing to special directed graph classes, in particular bidirected graphs.**

I. Introduction

Waypoint routing is a fundamental communication model in which packets need to visit a sequence of waypoints along their route. Waypoint routing has many applications, e.g., related to security policies [1], [2], [3], [4], emerging network services such as service function chaining [5], [6], [7], [8], [9], or segment routing [10], [11], [12], [13].

For example, computer networks today consist of a large number of so-called middleboxes (in the order of the number of routers [1]) providing various functionality inside the networks, related to security (e.g., firewalls, NATs) and performance (e.g., proxies, traffic optimizers). In order to benefit from (or enforce) these middleboxes, traffic needs to be steered through the functions ("waypoints") explicitly, as in Fig. 1. This is non-trivial especially in virtualized environments and in the context of Network Function Virtualization (NFV), where virtualized middleboxes can be deployed more flexibly. Software-Defined Networking (SDN) is a particularly useful technology in this context, as it facilitates the definition of such more flexible routes.

This paper is concerned with the algorithmic aspects underlying waypoint routing. Interestingly, only little is known today about the algorithmic problems, besides that the problem is typically hard on general network topologies [14].

ISBN 978-3-903176-08-9

Our paper is motivated by the fact that real-world networks (e.g., datacenter, enterprise, carrier networks) are often not general or "worst-case" but feature additional structure, which can potentially be exploited toward more efficient algorithms. Accordingly, we initiate in this paper the study of waypoint routing on specific network topologies.

A. Our Contributions

This paper studies the problem of computing (shortest) paths through an arbitrary number of waypoints on special network families. Our main contribution is a, in some sense, tight characterization of the network topologies on which routes through waypoints can be computed in polynomial time. Concretely, we provide an algorithm to compute waypoint routes on the important graph family of outerplanar graphs (which are of treewidth at most two). We show that it is difficult to go significantly beyond the graph families studied above, by deriving NP-hardness results on slightly more general graph families already (graphs of treewidth three). We also provide a polynomial algorithm for shortest routes on any constant treewidth, as long as the number of waypoints is also constant, with the added feature that the flow-sizes may change after each waypoint traversal. Additionally, we present various algorithmic and complexity results on special directed graphs, in particular on special bidirected graphs such as so-called cactus topologies.

B. State-of-the-Art and Novelty

The recent article by Amiri et al. [14] provided a first chart for this waypoint routing problem in general graph classes. Their focus is on providing intractability results and methods for few waypoints, but they present no algorithms to handle an arbitrary number of waypoints beyond trees and DAGs.

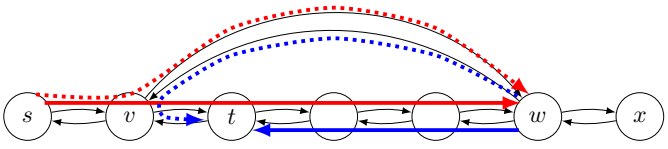


Fig. 1. In this introductory example, the task is to route the flow of traffic from the source s to the destination t via the waypoint w. When routing via the solid red (s, w) path, followed by the solid blue (w, t) path, the combined walk length is $5 + 3 = 8$. A shorter solution exists via the dotted red and blue paths, resulting in a combined walk length of $2 + 2 = 4$. Observe that when the waypoint would be on the node x, no node-disjoint path can route from s to t via the waypoint. Furthermore, some combinations can violate unit capacity constraints, e.g., combining the solid red with the dotted blue path induces a double utilization of the link from v to t.

# Waypoints	Feasible Algorithms	Known Hardness	Demand Change Optimal Algorithms	Demand Change Hardness
Arbitrary	**P**: Outerplanar ($\texttt{tw} \leq 2$) Corollary 2	**Strongly NPC**: $\texttt{tw} \leq 3$ Theorem 4	**P**: Tree (equivalent to $\texttt{tw}$ of 1) [14]	**NPC**: Unicyclic ($\texttt{tw} \leq 2$) Theorem 4
Constant	**P**: General graphs [14]	**P**: General graphs [14]	**P**: Constant treewidth $\texttt{tw} \in O(1)$ Theorem 3	**Strongly NPC**: General graphs [14]

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN SPECIAL UNDIRECTED GRAPHS.

The goal of this paper is to chart the algorithmic landscape of *special graph classes*, motivated by often highly structured computer networks. Our main results on undirected graphs are presented in Table I, but we also provide further new insights w.r.t. algorithms for special directed graphs, whereas [14] only provided NP-hardness results on general directed graphs.

C. Organization

The remainder of this paper is organized as follows. Section II introduces the problem and model more formally, along with studying the example of a single waypoint. Our in-depth algorithmic results are presented in Section III, whereas the complementing intractability proofs can be found in Section IV. We present further related work in Section V and conclude in Section VI.

II. THE PROBLEM AND MODEL

We study computer networks, modeled as connected undirected, directed, or bidirected [15] graphs $G = (V, E)$ with $|V| = n$ nodes (switches, middleboxes, routers) and $|E| = m$ links, where each link $e \in E$ has a capacity $c : E \to \mathbb{N}_{>0}$ and a weight (cost) $\chi : E \to \mathbb{N}_{>0}$. Bidirected graphs (also known as, e.g., Asynchronous Transfer Mode (ATM) networks [16] or symmetric digraphs [17]) are directed graphs with the property that if a link $e = (u, v)$ exists, there is also an anti-parallel link $e' = (v, u)$ with $c(e) = c(e')$ and $\chi(e) = \chi(e')$.

Given (1) a (bi/un)directed graph, (2) a source $s \in V$ and a destination $t \in V$, and (3) a set of k waypoints in V, the *waypoint routing problem* asks for a flow-route R (i.e., a walk) from s to t that (i) visits all waypoints in $\mathscr{W}$ and (ii) respects all link capacities. Without loss of generality, we normalize link capacities to the size of the traffic flow, removing links of insufficient capacity. Unless specified otherwise, we will assume at most one waypoint per node, though it may be that $s = t$. Waypoints may also change the traffic rate, where the demand can be denoted as follows: from s to w_1 by d_0, from w_1 to w_2 by d_1, etc. That said, if not stated explicitly otherwise, we will assume that $d_0 = d_1 = \ldots = d_k = 1$, and refer to this scenario as *flow-conserving*.

The waypoints depend on each other and must be traversed in a pre-determined order: every waypoint w_i may be visited at any time in the walk, and as often as desired (while respecting link capacities), but the route R must contain a given ordered node sequence $s, w_1, w_2, \ldots, w_k, t$. For example, in a network with stringent dependability requirements, it makes sense to first route a packet through a fast firewall before performing a deeper (and more costly) packet inspection.

We are interested both in *feasible* solutions (respecting capacity constraints) as well as in *optimal* solutions. In the context of the latter, we aim to optimize the cost $|R|$ of the route R, i.e., we want to minimize the sum of the weights of all traversed links.

Lastly, for ease of reference, we might denote the undirected waypoint routing problem by WRP, the directed version by DWRP, and the bidirected version by BWRP.

Before directly presenting our algorithms and complexity results, we start with a warm-up, considering the case of a single waypoint in bidirected networks.

A. An Introductory Case Study: A Single Waypoint

We first examine the case of a *single waypoint* w, which requires finding a shortest $s - t$ route through this waypoint. Amiri et al. [14] already 1) provided a polynomial-time algorithm for undirected graphs and 2) showed the NP-hardness for directed graphs. We thus complement their results by providing an algorithm for bidirected graphs as an introduction.

One waypoint: greedy is optimal. Simply taking two *shortest paths* (SPs) $P_1 = SP(s, w)$ and $P_2 = SP(w, t)$ in a greedy fashion is sufficient, i.e., the route $R = P_1 P_2$ is always feasible (and thus, also always optimal in regards to total weight).

Suppose this is not the case, that is, $P_1 \cap P_2 \neq \emptyset$, possibly violating capacity constraints. Among all nodes in $P_1 \cap P_2$, let u and v be, resp., the first and the last nodes w.r.t. to the order of visits in R. Let P_i^{xy} denote the sub-path connecting x to y in P_i. Thereby we have $R = P_1 P_2 = P_1^{su} P_1^{uv} P_1^{vw} P_2^{wu} P_2^{uv} P_2^{vt}$ (Fig. 2). Let $\bar{\mathcal{P}}$ be the reverse of any walk $\mathcal{P}$ obtained by replacing each link $(x, y) \in \mathcal{P}$ with its anti-parallel link (y, x). Observe that for $P_1' = P_1^{su} \bar{P}_2^{wu}$ and $P_2' = \bar{P}_1^{vw} P_2^{vt}$ we have that P_1' is at most as long as P_1 (because P_1 is shortest) and P_2' is shorter than P_2 (by P^{uv}), a contradiction to P_2 being a shortest path.

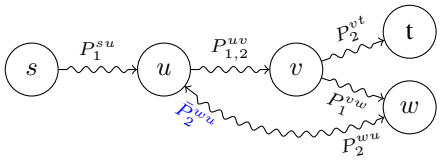


Fig. 2. The directed path from u to v is traversed two times in R.

Two waypoints: can be infeasible! While we saw that it is always possible to route through a single waypoint in bidirected graphs, already two waypoints can prevent a valid solution.

In the example of Figure 3, an $s-t$ route traversing first w_1 and w_2 second must use the link from w_2 to w_1 twice. Hence, the feasibility of a solution depends on the link capacity.

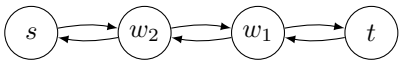


Fig. 3. In this unit capacity network, the task is to route the flow of traffic from s to w_1, then to w_2, and lastly to t. To this end, the link from w_2 to w_1 must be used twice.

After this brief introduction, we next study algorithms and complexity beyond the simple case of a single waypoint.

III. Exploring Computational Tractability

Computer networks often have very specific structures: for example, many data centers are highly structured (e.g., are based on Clos topologies [18]), but also enterprise and router-level AS topologies for example, while being less symmetric, often come with specific properties (e.g., are sparse). In this light, the general hardness results provided in [14] may be too pessimistic: in practice, much faster algorithms may be possible which are tailored toward and leverage the specific network structure. For example, as already pointed out in [14], the waypoint routing problem can be solved quickly on undirected tree or DAG topologies

Accordingly, in this section we explore the waypoint routing problem on specific graph families. In particular, we are interested in sparse graphs. We conducted a small empirical study using Rocketfuel topologies [19] and Internet Topology Zoo graphs [20], and found that they often have a low path diversity: almost half of these graphs are *outerplanar*, and one third are *cactus graphs*:

- a graph is outerplanar if it has a planar drawing s.t. all vertices are on the outer face of the drawing [21]
- a graph is a cactus graph if any two simple cycles share at most one node [22] (every cactus graph is outerplanar)

A. General Observations and Reductions

As first pointed out in [14] for undirected graphs, there is a direct algorithmic connection from the link-disjoint path problem to BWRP with unit capacities. By setting $s_1 = s$, $t_1 = w_1$, $s_2 = w_1$, $t_2 = w_2$, ..., a $k+1$ link-disjoint path algorithm also solves unit capacity BWRP for k waypoints. This method can be extended to general capacities via a standard technique, by replacing each link of capacity $c(e)$ with $\lfloor c(e) \rfloor$ parallel links of unit capacity and identical weight.

Hence, we can apply the algorithm from Jarry and Prennes [17], which solves the feasibility of the link-disjoint path problem on bidirected unit capacity multigraphs for a constant number of paths in polynomial runtime.

Theorem 1: Let $k \in \mathcal{O}(1)$. Feasible solutions for BWRP can be computed in polynomial time.

The optimal solution already for few link-disjoint paths still puzzles researchers on bidirected graphs, but the problem seems to be non-trivial on undirected graphs as well: while feasibility for a constant number of link-disjoint paths is polynomial in the undirected case as well [23],[24], optimal algorithms for 3 or more link-disjoint paths are not known, and even for 2 paths the best result is a recent randomized high-order polynomial-time algorithm [25]. For directed graphs, already 2 link-disjoint paths pose an NP-hard problem [26].

Furthermore, leveraging our connection to disjoint path problems again, we can also make the following observation, which we will use for special directed graphs and a non-constant amount of waypoints on some undirected graphs.

Observation 1: For any graph family on which the $k+1$ disjoint paths problem is polynomial-time solvable, we can also find a route through k waypoints in polynomial time on graphs of unit link capacity.

Thus, it immediately follows from [27] that the single waypoint routing problem is polynomial time solvable on semicomplete directed graphs, where a directed graph is called semicomplete, if there is at least one directed link between every pair of nodes.

Another case are directed graphs with constant independence number α, where $\alpha = \alpha(G)$ denotes the maximum size of an independent set in G. Then, for constant $\alpha, k \in O(1)$, a polynomial time DWRP algorithm exists, using [28].

Having a well-connected graph helps as well: On random undirected graphs G, where the set of $2k$ endpoints are chosen by an adversary (e.g., to compute a waypoint routing), it holds with high probability that the k paths exists, if $k \in O(n/\log n)$ and the minimum degree of G is some sufficiently large constant. The paths can be constructed in randomized time of $O(n^3)$ [29]. Similar results also hold on Expander graphs [30].

B. Algorithms: Parametrized by Treewidth I/II

For a further example, on bounded treewidth graphs, and as long as the number of waypoints k is logarithmically bounded, the problem is polynomial time solvable, because the link-disjoint paths problem is polynomial time solvable.

We briefly introduce the notion of treewidth as in [31], with alternate analogous descriptions and further examples provided in, e.g., Bodlaender and Kloks in [32], [33], [34]: Given an undirected graph $G = (V, E)$, a *tree decomposition* $\mathcal{T} = (T, X)$ of G is a bijection between a collection X and a tree T, s.t. every element of X is a set of nodes from V with: 1) each graph node is contained in at least one tree node, which is in turn called a *bag* (separator), 2) the tree nodes containing a node v form a connected subtree of T, and 3) nodes are adjacent in the graph only when the corresponding subtrees have a node in common. The *width* of $\mathcal{T} = (T, X)$ is the number of elements in the largest set in X minus 1. The *treewidth* $\mathtt{tw}$ is the minimum width over all tree decompositions of G. We will make use of these definitions again in Section III-D.

For a treewidth decomposition of width $\leq \mathtt{tw}$ and k link-disjoint paths, Zhou et al. [35] provide an algorithm with a runtime of

$$O\left(n((k+\mathtt{tw}^2)k^{\mathtt{tw}(\mathtt{tw}+1)/2} + k(\mathtt{tw}+4)^{2(\mathtt{tw}+4)k+3}\right) . \quad (1)$$

As a constant-factor approximation of treewidth decompositions can be obtained in polynomial time [36], also beyond constant

treewidth, it is therefore possible to solve the waypoint routing problem for any values of t and k s.t. Equation (1) stays polynomial. E.g., $\mathtt{tw}, k \in O(\sqrt{\log n / \log\log n})$, due to $f(n)^{g(n)} = \exp(\ln(f(n)^{g(n)})) = \exp(g(n)\ln(f(n)))$. This idea can also be extended to polylogarithmic functions $f(n), g(n) \in \text{polylog}(n)$, obtaining quasi-polynomial runtimes of $2^{\text{polylog}(n)} \in \text{QP}$. Quasi-polynomial algorithms fit sort of in between polynomial and exponential algorithms and it is widely believed that NP-complete problems are not in QP [37].

Unit capacities can be modeled by introducing parallel links and in particular subdividing them by placing auxiliary nodes in the center. For each such new path of length three, we can add the three nodes of the path to a new bag, and connect it to the original bag. Unless the graph is a tree (in which case the treewidth increases by one), the treewidth remains unchanged.

We thus obtain the following corollary, which does not find shortest routes and is not applicable to demand changes:

Corollary 1: In undirected graphs with a treewidth of $\mathtt{tw}$ and k waypoints, we can solve the waypoint routing problem in polynomial time for the following combinations:

- Constant $\mathtt{tw} \in O(1)$, logarithmic $k \in O(\log n)$
- $\mathtt{tw} \in O(\sqrt{\log n})$, constant $k \in O(1)$
- $\mathtt{tw}, k \in O\left(\sqrt{\log n / \log\log n}\right)$.

In quasi-polynomial time, we can solve:

- $\mathtt{tw}, k \in \text{polylog}(n)$.

Nonetheless, note that the non-parallel unit capacity observation is of limited use in general: for a negative example, an outerplanar graph requires nodes to touch the outer face, however, this property will be lost during the graph transformation. Yet, as we will show in the following, solutions for outerplanar graph exist, even in arbitrarily capacitated networks. We note that outerplanar graphs have a treewidth of $\mathtt{tw} \leq 2$.

C. Algorithms: Outerplanar and Cactus Graphs

Undirected Outerplanar Graphs. We first prove the following lemma, which we then use for outerplanar graphs.

Lemma 1: Let $\mathcal{I}$ be the class of undirected WRP with

1) the graph G is planar (w.l.o.g. we have a planar drawing),
2) the maximum capacity is $c_{\max}$, w.l.o.g. $n \geq c_{\max} \in \mathbb{N}$,
3) s, t and all waypoints touch the outer face $\mathcal{F}$ of G,
4) for every node $v \notin \mathcal{F}$, $\Sigma_{e:\,\{u,v\} \in E(G)} c(e)$ is even.

Then the *feasibility* of the ordered waypoint routing problem in the class $\mathcal{I}$ is decidable in time $O(n^2)$, and the construction of a feasible solution taking time $O(n^2 \cdot c_{\max}^2)$.

Proof: Let $I \in \mathcal{I}$ be an instance of the problem. Suppose s, t are the source and terminal and $w_1, \ldots, w_k$ are waypoints. Define $w_0 = s, w_{k+1} = t$. We construct an equivalent instance of the link-disjoint paths problem as follows. Replace each link $e = \{u, v\}$ with capacity c by $c \leq c_{\max}$ links with capacity 1, then subdivide those links once, i.e., the number of nodes is in $O(m \cdot c_{\max})$. In the newly created instance of link-disjoint paths problem:

1) The input graph is planar,
2) all terminal pairs touch the outer face,
3) the degree of every node not on the outer face is even.

If only condition 1) and 2) hold, the problem is NP-hard [38]. But for this class of link-disjoint paths problems, there are polynomial time algorithms [39] with the following properties: Let b be the number of nodes on the outer face and n' be the total number of nodes. Because the graph is planar we have $m = O(n)$ and $n' = O(n)$. The feasibility of the link-disjoint path problem can be tested in $O(bn')$ and constructing the paths can be done in $O(n'^2)$ which gives us the desired polynomial time solutions for the original problem. ■

This directly implies the following result:

Corollary 2: In undirected outerplanar graphs with a maximum link capacity of $c_{\max}$, the waypoint routing problem is decidable in time $O(n^2)$, with an explicit construction obtainable in time $O\left(n^2 \cdot \min\left\{n^2, c_{\max}^2\right\}\right)$.

A solution to the shortest waypoint routing problem cannot be obtained via the same reduction: Brandes et al. [40] showed the minimum total length link-disjoint path problem to be NP-hard on graphs satisfying the three conditions mentioned above, already when the maximum degree is at most 4.

For bidirected cactus graphs of constant capacity, the ordered waypoint routing problem can be optimally solved in polynomial time, as we show next.

Bidirected Cactus Graphs The difficulty of BWRP lies in the fact that the routing from w_i to w_{i+1} can be done along multiple paths, each of which could congest other waypoint connections. Hence, it is easy to solve BWRP optimally (or check for infeasibility) on trees, as each path connecting two successive waypoints is unique.

Lemma 2: BWRP can be solved optimally in polynomial time on trees.

For multiple path options, the problem turns NP-hard though (Theorem 6). To understand the impact of already two options, we follow-up by studying rings.

Lemma 3: BWRP is optimally solvable in polynomial time on bidirected ring graphs where for at least one link e holds: $c(e) \in \mathcal{O}(1)$.

Proof: We begin our proof with $c(e) = c(e') = 1$. Observe that every routing between two successive waypoints has two path options P, clockwise or counter-clockwise. We assign one arbitrary path P_e to traverse e, and another arbitrary path $P_{e'}$ to traverse e'. By removing the fully utilized e and e', the remaining graph is a tree with two leaves, where all routing is fixed, cf. Lemma 2.

We now count the path assignment possibilities for e, e': by also counting the "empty assignment", we have at most $(n+1)n$ options, where the optimal routing immediately follows for each option. For these $\mathcal{O}(n^2)$ possibilities, we pick the shortest feasible one. I.e., BWRP can be solved optimally in polynomial time on rings with unit capacity. To extend the proof to constant capacities $c(e) \in \mathcal{O}(1)$, we use an analogous argument, the number of options for assignments to e and e' are now $\mathcal{O}\left(n^{2c(e)}\right) \in \text{P}$. Thus, the lemma statement holds. ■

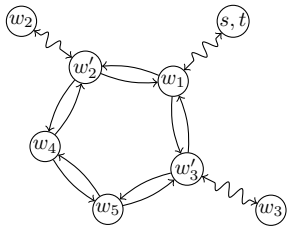


Fig. 4. In this cactus graph, we illustrate the algorithm of Theorem 2 w.r.t. the permutation $w_1 w_2 w_3 w_4 w_5$.

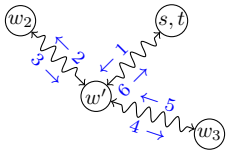


Fig. 5. Once the ring links are contracted, w' replaces the whole ring. Consequently, the permutation reduces to $w' w_2 w_3$. The sub-routes are numbered sequentially.

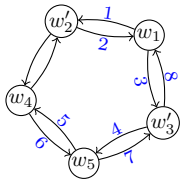


Fig. 6. The permutation induced on the ring is $w_1 w_2' w_3' w_4 w_5$. In the sub-problem, we have $s = t = w_1$. The numbers represent the order of node traversal in the optimal route.

We now focus on the important case of cactus networks. As mentioned earlier, our empirical study using the Internet Topology Zoo[1] data set shows that one third are *cactus graphs*.

Theorem 2: BWRP is optimally solvable in polynomial time on cactus graphs with constant capacity.

Proof: The idea is to 1) shrink the cactus graph down to a tree, 2) see if for the relevant subset of waypoints (to be described shortly) the feasibility holds on that tree, 3) reincorporate the excluded rings and find the optimal choice of path segments within each ring, and 4) construct an optimal route by stitching together the sub-routes obtained from the tree and the segments from each ring.

Let $\mathcal{C}$ be the cactus graph (Fig. 4) and $T_{\mathcal{C}}$ be the tree obtained after contracting all the links on each rings. As a result of this link contraction, those waypoints previously residing on rings are now replaced by new (super) waypoints in $T_{\mathcal{C}}$ (Fig. 5). Each super node represents either a subtree of adjacent rings or just an isolated ring. Let $\mathscr{W}'$ denote the waypoints in $T_{\mathcal{C}}$. Observe that any feasible route in $\mathcal{C}$ through $\mathscr{W}$ corresponds to one unique feasible route in $T_{\mathcal{C}}$ through nodes in $\mathscr{W}'$. Next, we show that either the feasible route in $T_{\mathcal{C}}$ (if exists) can be expanded to an optimal route for $\mathcal{C}$, or there is no feasible route in $\mathcal{C}$ at all. If $T_{\mathcal{C}}$ is not feasible then we are done. Otherwise, let R be the (unique) route in this tree. For each ring, R induces some *endpoints* (Fig. 6), one endpoint on each node that is either a) the joint of $T_{\mathcal{C}}$ and the ring, or b) the joint with its adjacent rings. Now we focus on the subproblem induced by this ring and the new waypoint set $\mathscr{W}''$ (to be specified) as follows.

For each endpoint that is visited by R add a waypoint to $\mathscr{W}''$. Then, using the algorithm described in the proof of Lemma 3, find an optimal route R_{ring} visiting all the nodes in $\mathscr{W}''$ respecting the order imposed by R. If no such route exists, the instance is not feasible. Otherwise, remove from R every occurrence of the super node that represents this ring to get a disconnected route. For each missing part, reconnect the endpoints using the segment of R_{ring} restricted to these endpoints. Repeat this for every ring; denote the resulting route as R'.

Finally, we argue that R' is optimal. This is the case because its pieces were taken from sets of sub-routes, where each set, covers a disjoint–or more precisely, node-disjoint up to endpoints–component of $\mathcal{C}$. Moreover, the set of sub-routes taken from an individual (disjoint) component (i.e. tree or ring) is optimal on that component. Therefore the total length is optimal. ■

We next turn our attention to graphs of constant treewidth.

D. Algorithms: Parametrized by Treewidth II/II

Let us quickly recap the results on undirected graphs of bounded treewidth $\mathtt{tw}$ found so far:

1) For constant $\mathtt{tw}$, we can compute walks for $k \in O(\log n)$ waypoints, but those walks will not be optimal (shortest) and the flow has to be of unit size. The same holds for outerplanar graphs (a class with $\mathtt{tw} = 2$) for $k \in O(n)$.
2) For $\mathtt{tw} = 1$ ($\equiv$ trees), one can compute shortest walks with demand changes, even for $k \in O(n)$ [14].

As pointed out in the beginning of Section III, many network topologies have low treewidth, especially in the wide-area and enterprise context (e.g., the Rocketfuel and Topology Zoo networks [19]). We now tackle a problem we thus deem to be realistic: in practice, the number of waypoints visited by a given flow is likely to be a small constant.

Theorem 3: In undirected graphs with bounded treewidth $\mathtt{tw} \in O(1)$ and a fixed number $k \in O(1)$ of waypoints, we can solve the shortest waypoint routing problem with demand changes in a runtime of $O(n)$.

Proof: Our proof will be via dynamic programming of a *nice tree decomposition* [41] $\mathcal{T} = (T, X)$ of G. Using the ideas and terminology of Kloks [34], a tree decomposition is nice if each bag of $\mathcal{T}$ is either a *leaf bag*, a *forget bag* (one node is removed from the separator), an *introduce bag* (a node is added), or a *join bag* (its two children q_1, q_2 contain the same nodes). For bags b, we thus define signatures σ_b, representing already computed solutions of b, such that by dynamically programming $\mathcal{T}$ bottom-up, we obtain an optimal walk $\mathcal{W}$ at the root bag of $\mathcal{T}$, if such a $\mathcal{W}$ exists.

In every optimal solution $\mathcal{W}$, each path from a w_i to a w_{i+1} will cross each separator b of G at most $\mathtt{tw}$ times. Due to optimality, these individual paths will traverse every node at most once. Hence, a signature σ_b only needs to represent the at most $k \cdot \mathtt{tw}$ crossings (endpoints) of partial paths through the subgraph of b, and the link utilizations these paths use in $E(b)$. We additionally store if a path, for from w_i to w_{i+1}, with only one endpoint in the signature, contains either w_i

[1] See http://www.topology-zoo.org/.

or w_{i+1}. Note that at most one such path each will exist at any time due to optimality. Due to $k, \mathtt{tw} \in O(1)$, we have only $O(1)$ different possible signatures for each bag b, with each signature containing only $O(1)$ elements. As common, we assume that we can perform standard operations (additions, comparisons etc.) of numerical values in constant time, else, an extra logarithmic factor needs to be included in the total runtime. We now present the required algorithms for the induction.

- **Leaf bags b:** In constant time, we can generate all valid signatures, containing at most k paths (each without any links). The only restriction is that if $v \in V(b)$ is a waypoint w_i, its paths to w_{i-1} and w_{i+1} must exist.
- **Forget bags b:** Let v be the node s.t. for the child q of b holds: $V(q) \setminus \{v\} = V(b)$. If v is not a waypoint, then the valid signatures of b are exactly those of q which do not use v as endpoints. If v is a waypoint w_i, then additionally must hold: v must be an endpoint of a path from w_{i-1} and the endpoint of a path to w_{i+1}.
- **Join bags b:** We first 1) describe the program and then 2) prove its correctness. 1): Given two valid signatures of b's children q_1, q_2, we perform all possible concatenations, of endpoints of paths for the same w_i to w_{i+1}, at the separator nodes $V(b)$, checking a) that the union of the link utilizations in $E(b)$ respect the link capacities and b) that no loops are created (we know the endpoints of each (sub-)path and the their link utilizations in $E(b)$, if they share a link outside $E(b)$, a signature of minimum size will not), which results in valid signatures σ_b of b. 2): Assume we missed some valid signature σ_b of b: Given σ_b, we split the paths across the separator, resulting in valid signatures $\sigma_{q_1}, \sigma_{q_2}$ and their subpaths, a contradiction. For an illustration of this procedure, we refer to Figure 7.
- **Introduce bags b:** Again, we first 1) describe the algorithm and then 2) prove its correctness. 1): For each signature σ_q of the child q of b, where $V(q) \cup \{v\} = V(b)$, we first generate all possible combinations of empty paths at v. Then, we distribute the link set of $E(b)$ over the endpoints in all possible variations, checking if each distribution can generate some valid signature by possibly moving the endpoints of the subwalks (and possibly, concatenating some). If the answer is yes, we also generate all possible signatures out of these distributions, again by allowing to move the endpoints and allowing to concatenate paths, always respecting capacity constraints. As we only handle $O(1)$ elements, we only perform $O(1)$ operations (covered below). 2): Again, assume we did not program some valid signature σ_b of b. We then obtain a valid signature of q by removing v, splitting all paths that traverse it into two, or, if they have v as an endpoint, cutting off v, or, if the path only contained v, by removing these paths. As the reverse operation will be performed by the prior algorithm, σ_b would have been obtained.

Each of the above programs be be run in a time of $O(1)$, assuming constant size $b, \mathtt{tw}, k \in O(1)$.

Furthermore, we implicitly assumed that for each signature, we also store a representative set of paths s.t. their total length is minimized. I.e., when generating signatures multiple times for introduce and join nodes, we only keep representatives of minimum total length. Hence, after dynamically programming the nice tree decomposition $\mathcal{T}$ bottom-up, we consider all solutions at the root node: If an optimal solution exists, it will be represented by a signature, and thus, we can choose a walk through the waypoints of minimum length.

It remains to prove the desired runtime of $O(n)$: For constant treewidth $\mathtt{tw} \in O(1)$, we can obtain a nice tree decomposition of width $O(\mathtt{tw})$ with $O(n)$ bags in a runtime of $O(n)$ using the methods from [34], [36]. As the dynamic program requires time $O(1)$ for each of the $O(n)$ bags, and as each of the $O(1)$ possible solutions can be checked in time $O(n)$, the claim follows. ∎

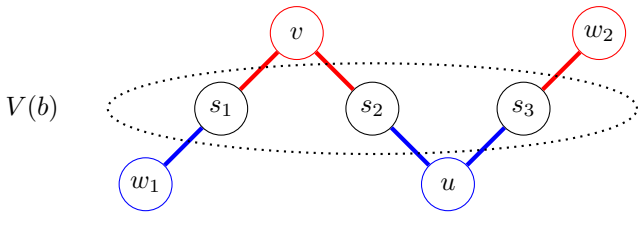


Fig. 7. In this example, the separator is shown in the middle, containing the nodes $V(b) = V(q_1) = V(q_2) = \{s_1, s_2, s_3\}$. By splitting the path from w_1 to w_2 along the separator, we obtain multiple paths per side, their number being bounded by the size of the separator. Observe that when two sub-paths, between the same set of waypoints, share a node, this node must be an endpoint for both; otherwise, minimality is violated.

IV. Hardness

In the previous Section III we presented various polynomial-time algorithms for undirected and directed graphs. In this section we present complementing hardness results, to clarify the corresponding intractability bounds. In comparison, previous work [14] provided NP-hardness results for general graphs, leaving the finer details where the border lays between polynomial-time algorithms and intractability to future work.

We begin by studying the treewidth of undirected graphs in Section IV-A, followed by the NP-hardness on (un)directed unicyclic graphs under flow-size changes in Section IV-B. Lastly, we investigate general bidirected graphs in Section IV-C, where hardness already strikes without flow-size changes, as in Section IV-A on undirected graphs.

A. Hardness: Parametrized by Treewidth

We have shown that for a large graph family of treewidth at most 2, the outerplanar graphs (which also include cactus graphs for example), the routing paths can be computed efficiently on undirected graphs. This raises the question whether the problem can be solved also on graphs of treewidth larger than 2, or at least for *all* graphs of treewidth at most 2. While the latter remains an open question, in the following we show that problems on graphs of treewidth 3 (namely series-parallel graphs with an additional node connected to all other nodes) are already NP-hard in general.

Theorem 4: The problem of routing through an arbitrary number of waypoints is strongly NP-complete on undirected graphs of treewidth at most 3.

Proof: We reduce the ordered waypoint routing problem in graphs of treewidth at most 3 from the link-disjoint paths problem in series-parallel graphs, the latter being strongly NP-complete [42].

Let I be an instance of the link-disjoint paths problem in a series parallel graph G with terminal pairs $T_I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. We construct a new instance $\mathcal{I}$ of the ordered waypoint problem as follows. Create a graph $G' := G$, then add one new node v to G' and links $\{t_i, v\}, \{s_j, v\}$ for $i, j \in [k], j \neq 1, i \neq k$.

For simplicity, set for now $s := s_1$, $w_1 := t_1$, $w_2 := v$, $w_3 := s_2$, $w_4 := t_2$, $w_5 := v$, $\ldots$, $t := t_k$, i.e., the order of waypoints is $s_1, t_1, v, \ldots, v, s_i, t_i, v, s_{i+1}, t_{i+1}, v, \ldots, t_k$, with $3k-2$ waypoints in total. I.e., v "hosts" $k-1$ waypoints, with a degree of $2(k-1)$. We will show later in the proof how to ensure at most one waypoint per node.

Claim: In any solution for $\mathcal{I}$, the union of the $k-1$ link-disjoint walks from s_i via v to t_{i+1} occupy all links incident to v.

Proof: Any walk from s_i via v to t_{i+1} must leave and enter v, using two links. Hence, the union of all these $k-1$ link-disjoint walks occupy all $2k-2$ links incident to v. □

We can now prove the theorem: If I is a yes-instance, then $\mathcal{I}$ is a yes-instance as well: We take the k s_i, t_i-paths from I, connect them in index-order with the $k-1$ paths t_i, v, s_{i+1}, and obtain the desired ordered waypoint routing.

It is left to show that if $\mathcal{I}$ is a yes-instance, then I is a yes-instance as well: Let $\mathcal{I}$ be a yes-instance. Define the path from s_i to t_i as in $\mathcal{I}$. As these paths do not use v or any of the links adjacent to it (otherwise the capacity of one of these links would be exceeded), these paths show that I is a yes-instance.

On the other hand, the treewidth of G' is at most the treewidth of G plus 1 (we can just put v in all bags of an optimal tree decomposition of G). To obtain at most one waypoint on v, we create $k-1$ cycles of length four, placing a waypoint on each, and merging another node with v. This construction does not increase the treewidth and also retains earlier proof arguments. As series-parallel graphs have a treewidth of at most 2 [43, Lemma 11.2.1], G' has a treewidth of at most 3. As the problem is clearly in NP, with the reduction being polynomial, the proof is complete. ■

We conjecture that it is possible to directly modify the proof presented in [42], to prove that the feasibility of the waypoint routing problem is hard even in series-parallel graphs.

B. Hardness: Flow-size changes and a single cycle

In case of non-flow conserving waypoints, NP-hardness strikes earlier already, namely on unicyclic graphs, which contain only one cycle, and thus have $\mathtt{tw} \leq 2$.

Theorem 5: On undirected unicyclic graphs in which waypoints are not flow-conserving, computing a route through $O(n)$ waypoints is weakly NP-complete, even if all waypoints can just increase (or, just decrease) the flow size by at most a constant factor.

Proof: Reduction from the weakly NP-complete PARTITION problem [44], where an instance I contains ℓ non-negative integers $i_1, \ldots, i_\ell$, $\sum_{j=1}^{\ell} i_j = S$, with the size of the binary representation of all integers polynomially bounded in ℓ.

We begin with the case that waypoints can change the flow size arbitrarily. W.l.o.g., let ℓ be even and $i_1 \leq i_2 \leq \cdots \leq i_\ell$. We create two stars (denoted left and right star) with $1 + \ell/2$ leaf nodes each, where all links have a capacity of S. We connect both star center nodes in a cycle, with the cycle links having a capacity of $S/2$ each, respectively.

Next, we place s, here also identified as w_1, on a leaf of the left star and t on a leaf in the right star. To distribute the remaining $\ell - 1$ waypoints $w_2, \ldots, w_\ell$, corresponding to the integers, we place the ones with even indices on leaves in the left star, and those with odd indices in the right star.

Suppose the routing starts with a size of i_1, is changed to i_2 by w_2 and so on. Then, solving the PARTITION instance I is equivalent to computing a waypoint routing, as the paths going along the cycle have to be partitioned into two sets, each having a combined demand of $S/2$.

So far, we assumed that waypoints can change the flow size arbitrarily – but hardness also holds if each waypoint can just increase (or, just decrease) the flow size by a constant amount. In order to do so, we replace the leaf nodes of the stars with paths of $O(\log S)$ waypoints, which are used to increase the demands to the desired size. ■

The directed graph case is analogous by putting all waypoints to one star, creating the same amount of intermediate dummy waypoints in the other star, which do not change the flow size, and replacing all undirected links with two directed links of opposite directions and identical capacity.

Corollary 3: On directed graphs, with the underlying undirected graph being unicyclic and where waypoints are not flow-conserving, computing a route through $O(n)$ waypoints is NP-complete, even if all waypoints can just increase (or, just decrease) the flow size by at most a constant factor.

For these two proofs, we used flow sizes that can be exponential in the graph size (binary encoded). Nonetheless, we refer to Table II, which shows that the problem also stays strongly NP-complete on general graphs.

C. Hardness: Bidirected graphs without flow-size changes

It follows from the earlier Corollary 3 that waypoint routing is already NP-hard on unicyclic bidirected graphs, when allowing flow-size changes. It remains to study NP-hardness in the case that the flow-size remains unchanged:

Theorem 6: Solving BWRP optimally is NP-hard.

Proof: Reduction from the NP-hard link-disjoint path problem on bidirected graphs $G = (V, E)$ [16]: given k source-destination node-pairs (s_i, t_i), $1 \leq i \leq k$, are there k corresponding pairwise link-disjoint paths?

For every such instance I, we create an instance I' of BWRP as follows, with all unit capacities: Set $s = s_1$ and $t = t_k$,

	# Waypoints	Feasible	Optimal	Demand Change Feasible	Optimal
Undirected	1	**P**		**Strongly NPC**	
	constant	**P**	?		
	arbitrary	**Strongly NPC**			
Directed	1	**Strongly NPC**			
	constant				
	arbitrary				

TABLE II
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS AS PROVIDED BY [14].

also setting waypoints as follows: $w_1 = t_1$, $w_3 = s_2$, $w_4 = t_2$, $w_6 = s_3$, $w_7 = t_3$, ..., $w_{3k-3} = s_k$. We also create the missing $k-1$ waypoints w_2, w_5, w_8, ..., w_{3k-4} as new nodes and connect them as follows, each time with bidirected links of weight γ: w_2 to $w_1 = t_1$ and $w_3 = s_2$, w_5 to $w_4 = t_2$ and $w_6 = s_3$, ..., w_{3k-4} to $w_{3k-3} = s_k$ and $w_{3k-5} = t_{k-1}$. I.e., we sequentially connect the end- and start-points of the paths.

Observe that BWRP is feasible on I' if I is feasible: We take the k link-disjoint paths from I and connect them via the $k-1$ new nodes in I'.

We now set γ to some arbitrarily high weight, e.g., $3k$ times the sum of all link weights. I.e., it is cheaper to traverse every link of I even $3k$ times rather than paying γ once. Thus, if I is feasible, the optimal solution of I' has a cost of less than $2 \cdot k \cdot \gamma$.

Assume I is not feasible, but that I' has a feasible solution R. Observe that a feasible solution of I' needs to traverse the $k-1$ new waypoints, i.e., has at least a cost of $2(k-1)\gamma$. As I was not feasible, we will now show that traversing every new waypoint $w_2, w_5, \ldots$ only once is not sufficient for a feasible solution of I'. Assume for contradiction that one traversal of $w_2, w_5, \ldots$ suffices: for each of those traversals of such a w_j, it holds that it must take place after traversing all waypoints with index smaller than j. Hence, we can show by induction that the removal of the links incident to the waypoints $w_2, w_5, \ldots$ from R contains a feasible solution for I. Thus, at least one of the waypoints $w_2, w_5, \ldots$ must be traversed twice, i.e., R has a cost of at least $2 \cdot k \cdot \gamma$.

We can now complete the polynomial reduction, by studying the cost (feasibility) of an optimal solution of I': if the cost is less than $2 \cdot k \cdot \gamma$, I is feasible, but if the cost is at least $2 \cdot k \cdot \gamma$ (or infeasible), I is not feasible. ∎

While many BWRP instances are not feasible (already in Figure 3), we conjecture that the feasibility of BWRP with arbitrarily many waypoints is NP-hard as well. This conjecture is supported by the fact that the analogous link-disjoint feasibility problems are NP-hard on undirected [44], directed [26], and bidirected graphs [16], also for undirected and directed ordered waypoint routing, see Table II.

V. RELATED WORK

While waypoint routing has recently received much attention in the literature, especially in the context of service function chaining [7], [9], [45], [46], we are not aware of any systematic study of the underlying algorithmic problem besides [14] which however does not consider special network families. We provide Table II for an overview of their results on general graphs.

In particular, our work is different from existing literature on the computation of routes through *unordered* waypoints [31]: the computation of shortest (link- and node-disjoint) paths and cycles through a *set* of k waypoints is a classic problem [47] which has traditionally been motivated by many different applications. Well-known results include, e.g., linear-time algorithms for $k = 3$ waypoints [26], [48] polynomial-time algorithms for constant k [24], polynomial-time deterministic algorithms to compute *feasible* paths for small $k = O((\log \log n)^{1/10})$, or a randomized algorithm (based on algebraic techniques) to compute a shortest simple cycle through a given set of k nodes or links in an n-node undirected network. These approaches however cannot be applied to compute routes through ordered waypoints.

Our work is also different from existing work which focuses on how to *admit* and allocate *multiple* walks, e.g., using randomized rounding and tolerating some capacity augmentation [49], [50], [51]. There are also extensions to more complex requests such as trees [51], [52]. In contrast, we in this paper focus on the allocation of a *single* walk, without violating capacity constraints.

Bibliographic Note. A first version of the results on bidirected graphs was presented at the Algocloud workshop [53].

VI. CONCLUSION

Waypoint routing is emerging as an important concept in various applications, however, the underlying algorithmic problem is not well-understood. With this paper, we have made a first step to put the waypoint routing problem into perspective. We presented a comprehensive characterization of the algorithmic complexity of the problem regarding the "special" network families which support a polynomial-time solution. In particular, we presented algorithms and hardness results for networks of different treewidth, and discussed implications of more directed networks. In our future work, we aim to investigate the implications of waypoint routing on specific applications, in particular, Traffic Engineering.

Acknowledgments. We like to thank Thore Husfeldt for inspiring discussions. Research partly supported by the Villum project ReNet as well as by Aalborg University's PreLytics project. Saeed Amiri's research was partly supported by the European Research Council (ERC) under the European Union's

Horizon 2020 research and innovation programme (grant agreement No 648527).

REFERENCES

[1] J. Sherry et al., "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012.

[2] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," in *Proc. ACM SIGMETRICS*, 2016.

[3] A. Matos, S. Sargento, and R. L. Aguiar, "Waypoint routing: A network layer privacy framework," in *Proc. IEEE GLOBECOM*, 2011.

[4] S. Ghorbani and B. Godfrey, "Towards correct network virtualization," in *Proc. SIGCOMM HotSDN*, 2014, pp. 109–114.

[5] P. Skoldstrom et al., "Towards unified programmability of cloud and carrier infrastructure," in *Proc. EWSDN*, 2014.

[6] ETSI GS NFV-IFA 003, "Network functions virtualisation (nfv); acceleration technologies; vswitch benchmarking and acceleration specification," in *Group Specification*, 2016.

[7] R. Soulé et al., "Merlin: A language for provisioning network resources," in *Proc. ACM CoNEXT*, 2014.

[8] P. S. et al., "Towards unified programmability of cloud and carrier infrastructure," in *Proc. EWSDN*, 2014.

[9] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proc. SIGCOMM*, 2015.

[10] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. GLOBECOM*, 2015.

[11] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *Proc. IEEE INFOCOM*, 2015, pp. 657–665.

[12] C. Filsfils et al., "Segment routing architecture," in *Internet draft*, 2014.

[13] C. Filsfils, P. Francois, S. Previdi, B. Decraene, S. Litkowski, M. Horneffer, I. Milojevic, R. Shakir, S. Ytti, W. Henderickx, J. Tantsura, S. Kini, and E. Crabbe, "Segment routing architecture," in *Segment Routing Use Cases, IETF Internet-Draft*, 2014.

[14] S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "Charting the Complexity Landscape of Waypoint Routing," *SIGCOMM Comput. Commun. Rev*, vol. 48, no. 1, January 2018.

[15] J. Edmonds and E. Johnson, "Matching: a well-solved class of linear programs," in *Combinatorial Structures and their Applications: Proceedings of the Calgary Symposium*. New York: Gordon and Breach, 1970, pp. 88–92.

[16] P. Chanas, "Reseaux atm: conception et optimisation," Ph.D. dissertation, University of Grenoble, 1998, these de doctorat dirige par Finke, Gerd et Burlet, Michel Sciences appliques Grenoble 1 1998. [Online]. Available: http://www.theses.fr/1998GRE10113

[17] A. Jarry and S. Pérennes, "Disjoint paths in symmetric digraphs," *Discrete Applied Mathematics*, vol. 157, no. 1, pp. 90–97, 2009.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[19] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[20] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.

[21] G. Chartrand and F. Harary, "Planar permutation graphs," *Annales de l'I.H.P. Probabilites et statistiques*, vol. 3, no. 4, pp. 433–438, 1967.

[22] D. Geller and B. Manvel, "Reconstruction of cacti," *Canad. J. Math*, vol. 21, pp. 1354–1360, 1969.

[23] K. Kawarabayashi, Y. Kobayashi, and B. A. Reed, "The disjoint paths problem in quadratic time," *J. Comb. Theory, Ser. B*, vol. 102, no. 2, pp. 424–435, 2012.

[24] N. Robertson and P. D. Seymour, "Graph Minors .XIII. The Disjoint Paths Problem," *J. Comb. Theory, Ser. B*, vol. 63, no. 1, pp. 65–110, 1995.

[25] A. Björklund and T. Husfeldt, "Shortest two disjoint paths in polynomial time," in *Proc. ICALP*, 2014.

[26] S. Fortune, J. E. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theor. Comput. Sci.*, vol. 10, pp. 111–121, 1980.

[27] J. Bang-Jensen, "Edge-disjoint in- and out-branchings in tournaments and related path problems," *J. Comb. Theory, Ser. B*, vol. 51, no. 1, pp. 1–23, 1991.

[28] A. O. Fradkin and P. D. Seymour, "Edge-disjoint paths in digraphs with bounded independence number," *J. Comb. Theory, Ser. B*, vol. 110, pp. 19–46, 2015.

[29] A. M. Frieze and L. Zhao, "Optimal construction of edge-disjoint paths in random regular graphs," *Combinatorics, Probability & Computing*, vol. 9, no. 3, pp. 241–263, 2000.

[30] A. M. Frieze, "Edge-disjoint paths in expander graphs," *SIAM J. Comput.*, vol. 30, no. 6, pp. 1790–1801, 2000.

[31] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "Walking Through Waypoints," in *LATIN*, ser. Lecture Notes in Computer Science, 2018.

[32] H. L. Bodlaender, "Treewidth: Structure and algorithms," in *SIROCCO*, ser. Lecture Notes in Computer Science, vol. 4474. Springer, 2007, pp. 11–25.

[33] ——, "A tourist guide through treewidth," *Acta Cybern.*, vol. 11, no. 1-2, pp. 1–21, 1993.

[34] T. Kloks, *Treewidth, Computations and Approximations*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 842.

[35] X. Zhou, S. Tamura, and T. Nishizeki, "Finding edge-disjoint paths in partial k-trees," *Algorithmica*, vol. 26, no. 1, pp. 3–30, 2000.

[36] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk, "An approximation algorithm for treewidth," in *Proc. FOCS*, 2013.

[37] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial Optimization*, ser. LNCS, vol. 2570. Springer, 2001, pp. 185–208.

[38] W. Schwärzler, "On the complexity of the planar edge-disjoint paths problem with terminals on the outer boundary," *Combinatorica*, vol. 29, no. 1, pp. 121–126, 2009.

[39] M. Becker and K. Mehlhorn, "Algorithms for routing in planar graphs," *Acta Informatica*, vol. 23, no. 2, pp. 163–176, 1986.

[40] U. Brandes, G. Neyer, and D. Wagner, "Edge-disjoint paths in planar graphs with short total length," 1996, Konstanzer Schriften in Mathematik und Informatik; 19.

[41] H. Bodlaender, "Dynamic programming on graphs with bounded treewidth," *Automata, Languages and Programming*, pp. 105–118, 1988.

[42] T. Nishizeki, J. Vygen, and X. Zhou, "The edge-disjoint paths problem is NP-complete for series–parallel graphs," *Discrete Appl. Math.*, vol. 115, 2001.

[43] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*. Philadelphia, PA, USA: SIAM, 1999.

[44] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[45] ETSI, "Network functions virtualisation – introductory white paper," *White Paper*, oct 2013.

[46] J. Napper, W. Haeffner, M. Stiemerling, D. R. Lopez, and J. Uttaro, "Service Function Chaining Use Cases in Mobile Networks," Internet-Draft, Apr. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06

[47] L. Perković and B. A. Reed, "An improved algorithm for finding tree decompositions of small width," *Int. J. Found. Comput. Sci.*, vol. 11, no. 3, pp. 365–371, 2000.

[48] H. Fleischner and G. J. Woeginger, "Detecting cycles through three fixed vertices in a graph," *Inf. Process. Lett.*, vol. 42, no. 1, pp. 29–33, 1992.

[49] G. Even, M. Medina, and B. Patt-Shamir, "Online path computation and function placement in sdns," in *Proc. SSS*, 2016.

[50] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *SIROCCO*, 2015.

[51] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in SDNs," in *SIROCCO*, 2016.

[52] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, "Minimum congestion mapping in a cloud," in *Proc. ACM PODC*, 2011.

[53] K.-T. Foerster, M. Parham, and S. Schmid, "A walk in the clouds: Routing through vnfs on bidirected networks," in *ALGOCLOUD*, ser. Lecture Notes in Computer Science, vol. 10739. Springer, 2017, pp. 11–26.

Complex Services Offloading in Opportunistic Networks

The An Binh Nguyen*, Marius Rettberg-Päplow*, Christian Meurisch†, Tobias Meuser*,
Björn Richerzhagen*, Ralf Steinmetz*
*Multimedia Communiations Lab (KOM), TU Darmstadt, Germany
Email: {firstname.lastname}@kom.tu-darmstadt.de, mberg@hotmail.de
†Telecooperation (TK), TU Darmstadt, Germany
Email: meurisch@tk.tu-darmstadt.de

***Abstract*—Situation awareness is important to plan relief work in emergency response. However, impaired communication and computation infrastructure makes it difficult to acquire and analyze information. Accordingly, complex and resource-intensive information processing can be offloaded through opportunistic ad hoc contact to, e.g., first responder mobile devices, leveraging their idle resources. Ensuring complete service execution without overloading individual devices is a challenging task in such dynamic networks. In this work, we propose handover mechanisms that utilize the current context of individual mobile devices to balance load and achieve complete task execution without requiring a global view on the opportunistic network. We study their scalability and performance by combining them with our unified message template for distributed service processing in the OMNeT++ simulation environment. The evaluation shows that our handover mechanisms increase the success rate significantly and achieve distributed load balancing.**

I. Introduction

In recent years, mobile services become more computation-intensive and more complex, requiring multiple processing stages and highly-specialized hardware. Executing such services on a single device is therefore impractical. To alleviate a device with limited computing capacity, a complex mobile service can be offloaded as a task to a remote cloud for execution. However, offloading to the cloud is not always possible due to overloaded or impaired infrastructures, which can occur, for instance, in emergency situations such as disaster scenarios. Opportunistic offloading [1] has been introduced as an emerging solution for offloading computation. Hereby, the computation tasks can be offloaded to a nearby stationary computing unit such as cloudlet [2], or to an opportunistic network formed by mobile devices [3]. While both approaches share the common idea of leveraging nearby available computing resources, offloading in an opportunistic network provides more flexibility and more advantages in favor of executing complex tasks with multiple processing stages. A complex task can be divided into several subtasks, and distributed to the participating mobile devices, leveraging their idle, and heterogeneous capabilities. Besides ensuring successful execution of the offloaded tasks, balancing services execution among the participating devices is also essential, and beneficial. On the one hand, load balancing relieves overloaded devices, effectively leading to improved overall performance. On the other hand, the energy consumption for executing the offloaded tasks by participating devices can be decreased through load balancing, resulting in (i) longer lifetime of opportunistic networks, which serves as communication medium during critical situations, and (ii) more acceptance of users to contribute their resources. Most approaches dealing with load balancing in mobile systems are based on global knowledge of the network to formulate the load balanced assignment as an optimization problem. In line with the dynamic nature of opportunistic networks, the optimization problem can also be solved in a distributed manner, as proposed in [4]. Still, rapidly changing environments require a flexible and adaptive approach to load balancing that takes changing conditions, resource constraints, heterogeneity of tasks and services, and mobility into account.

In this work, we propose several handover mechanisms for load balancing in complex services offloading based on the currently available context of single devices. To this end, we extend our previous work [5] on a task message template that allows the user to define a task and the services required to accomplish the defined task. Our message template bundles the control information and the corresponding payload data into a single message. This enables mobile devices to decide autonomously whether and how to participate in service processing. We implement our proposed mechanisms within the OMNeT++ simulator [6], allowing for an in-depth evaluation of their performance in terms of load balancing and success rate and their cost in terms of message overhead and latency.

In summary, the contributions of this paper are threefold:

- We propose several load balancing (*LB*) mechanisms for distributing complex tasks across devices in an opportunistic network, optimizing resource utilization and success rate, while minimizing the communication overhead.
- We develop a simulation environment based on OMNeT++, which integrates our earlier work on an adaptive task oriented message template [5].
- We conduct an extensive evaluation of our LB mechanisms within the OMNeT++ simulation environment. We show the overall performance gain, improved fairness, and inherited trade-offs of our proposed LB mechanisms.

ISBN 978-3-903176-08-9

The remainder of this paper is organized as follows. *First*, we discuss related work. *Second*, we give a brief introduction in our adaptive task message template (namely *ATMT*) for distributed in-network processing, and highlight an important open research challenge, namely, *distributed fair load balancing*. *Third*, we present our load balancing handover mechanisms and an in-depth evaluation relying on OMNeT++ simulations, before concluding the paper.

II. Related Work

Offloading computational workload in mobile systems, aiming to reduce network traffic have been studied in several research work. [7] propose a decentralized optimization model for the underlying operator placement problem. This approach, however, does not consider dynamic changes of the environment. Recent research on *Complex Event Processing (CEP)* in the context of vehicular networks has put more attention to adaptive mechanisms; an example is CEP operator migration [8]. Another research direction is edge computing, in which computation tasks are offloaded to nearby computing resources such as cloudet-upgraded router for processing [2]. In the aforementioned work, balancing computational workload is neglected.

Load balancing or fair resource allocation have always been an important research aspect in mobile networks. To analyze fair resource allocation, Fossati et al. [9] propose to extend the Jain's fairness index with a satisfaction factor of users. The problem of resource allocation is modeled through game theory, using their proposed metric. Tham et al. [4] target mobile edge networks and formulate a constrained optimization problem to achieve load balancing. The problem, however, has to be solved by a central entity. Fernando et al. [10] incorporate work stealing concepts in mobile crowd computing, allowing a worker device to take over workload from other devices. The authors focus on the practical implementation using mobile devices and, thus, do not consider work stealing in a large scale setup. Centralized coordination is impractical in an opportunistic network. Consequently, Benchi et al. [11] study the consensus problem in opportunistic networks, which allows each node to make a consent decision upon receiving enough votes from others. Comparable to our work is load balancing for services composition in opportunistic networks. Viswanathan et al. [12] use a time deadline for services composition to formulate an optimization problem, which can be solved by service providers in a distributed manner. The complexity of such optimization formulation is high, thus cannot cope well with the rapid changes of an opportunistic network. In [13], Sadid et al. introduce a hop by hop composition model designed for opportunistic networks, considering load and mobility of the devices. The authors propose to let each service provider decide on the next composition to cope with dynamic changes. Our work differentiates from [13] in that we explicitly incorporate uncertainty factors in our local optimization mechanisms to increase their robustness. Furthermore, we provide a thorough evaluation focusing on the quality of load balancing.

III. Scenario: In-Network Data Analysis in Emergency Situations

A. Scenario Description

To plan relief operations in emergency response situations efficiently, the relief workers need to have situational information. The required raw sensing data can be obtained through built-in sensors on the mobile devices as shown in [14]. Thereafter, these data have to be processed and analyzed to extract valuable information. A concrete example can be found in [15]. In this work, image processing techniques are applied to extract faces of victims through pictures shot by smart phones. To capture the situational overview, a large amount of data might be required. Processing all these data in a single device of the relief worker is inefficient. Two options are possible: (i) offloading the data analysis to cloud servers, (ii) offloading the data analysis to several surrogate devices for distributed processing. The first option is not always possible in case of impaired communication infrastructure, which often occurs in disaster situations. The second option provides a more flexible solution to analyze data, leveraging idle resources available in opportunistic network.

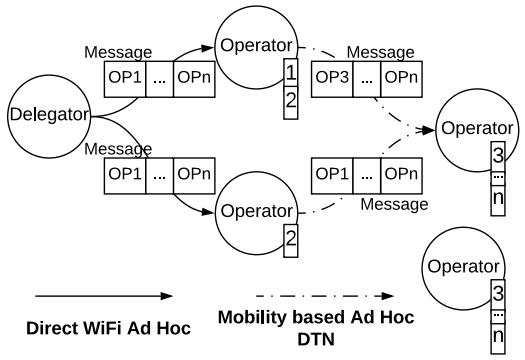


Fig. 1: Abstract system model of disseminating ATMT task messages in opportunistic networks for processing.

To facilitate distributed processing through mobile devices in the elaborated scenario, the *adaptive task-oriented message template* (ATMT) is proposed in [5]. The objective of this message template is to allow users to define an analysis goal and the operations/services required to accomplish this goal. Using the task message template, the data analysis is handed over from one device to the next device, wheres each device can perform one or several operations. Hence, the task is divided and processed in a distributed manner. The workflow of processing an ATMT message is illustrated in Figure 1. In this illustration, a device (called *delegator*) with required domain knowledge of how to process the data analyis, defines n operations ($op_1..op_n$) and disseminate the message into the opportunistic ad hoc networks. Each device participating in the processing (called *operator*) executes the operations provided by this device and hands over the processed message upon opportunistic contact with other devices for further executions.

B. Adaptive Task Message Template

The construction of the ATMT task message template designed in our previous work [5] is illustrated in Figure 2. To facilitate distributed processing in opportunistic networks, one of the objectives of ATMT is to allow the participating devices to cooperate without having to rely on any centralized coordination. Due to this reason, an ATMT message contains both meta-information required for processing and the belonging payload data. The meta-information is stored in the ATMT header, consisting of two parts, i.e., *message header* and *analysis header*. The first part is the fix-sized *message header*, which contains an UUID for identification, a checksum on the status of the processing and the length of the header. By comparing the checksum in the *message header*, a device can check on the current status of the processing and decides to merge, drop or to handover a task, without parsing the whole message content. The *analysis header* composes of an *operations graph* and a *data dictionary*. The *operations graph* is based on an acyclic directed graph, that is used to model the processing goal, the required operations/services, and the processing order. The *data dictionary* in the header maps the operations in the *operations graph* to the respective data pieces in the *ATMT payload*. When an operation is completed by a device, this device can replace the old payload data with the processed result. All in all, the construction of an ATMT message allows each device to make autonomous decision.

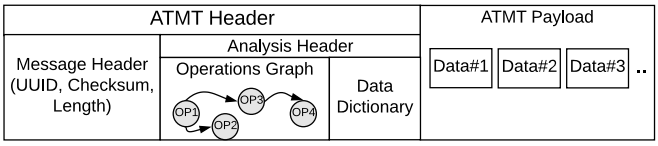


Fig. 2: Construction of ATMT task message template as designed in [5].

A system utilizing ATMT message to perform in-network data analysis as the aforementioned scenario depends on the heterogeneous capabilities of the devices, which can be translated into different roles. Four roles are conceived, i.e., *sensors* for obtaining raw data, *delegators* with the domain knowledge for constructing the *operations graph* which can be understood as a way to coordinate the devices in a distributed manner, *operator* for performing operations/services, and *forwarder* to handover the ATMT messages. As briefly described in the previous section, Figure 1 shows a sample workflow using ATMT concept. *Sensor* devices are omitted in the illustration. A *delegator* device receiving data from the sensors constructs an ATMT message, and hands over this message to its directly connected operators via WiFi ad hoc communication. Each operator processes the ATMT message and executes the operations/services required in the *operations graph* according to its available resource and services. The resulting ATMT messages can be forwarded through *store, carry and forward* concept of opportunistic mobile networks to another operator at later time for further processing. In doing so, the chances for successful execution of a complex analysis task can be increased.

IV. Challenges and Assumptions

Based on the description of the ATMT construction and the in-network data analysis workflow, we can identify several challenges. (i) A centralized coordination with the complete view over the services available in all mobile devices does not exist. Consequently, each device only has a partial view of the network. (ii) The devices considered in this work are highly dynamic and mobile. This requires adaptive mechanisms. (iii) Due to the challenges elaborated in (i) and (ii), the handover of ATMT messages in an uncoordinated way might lead to massive communication overhead and processing redundancies, i.e., workload waste. Optimizing both successful execution of complex ATMT tasks and load balancing under the aforementioned challenges is thus our main target.

With respect to the challenges and the elaborated application scenario, the following assumptions are made:

- Decentralized opportunistic ad hoc network: we focus on complex services offloading and distributed processing in an opportunistic ad hoc network. Thus, we assume that the devices are mobile and they are able to communicate if they are in WiFi range of each other.
- Heterogeneous resource and services: we assume that the participating devices possess different capabilities, i.e., each device has different resource capacity left, can provide different services, perform different operations.
- Cooperative behaviour: we assume that no participating device has malicious intention. To establish a trustworthy distributed processing environment in a mobile system, trust measurement concept such as in [16] can be utilized.
- Location-aware: we assume that each device is able to determine its own location.

V. Handover Mechanisms

We design our handover mechanisms with special focus on load balancing. Our target is to improve the distribution of workload among participating devices in an opportunistic network, taking into account the challenges and assumptions as previously discussed. According to Alakeel [17], we have to consider three main aspects when designing load balancing mechanisms for distributed systems, i.e., *transfer strategy, location strategy, information strategy*. *Transfer strategy* is the decision whether to offload/handover the task, *location strategy* indicates which destinations should the tasks be offloaded to, and *information strategy* refers to the context information which can be used to devise *transfer strategy* and *location strategy*. Accordingly, the *information strategy* is the most important component of handover mechanisms. W.r.t. our scenario, the *information strategy* is limited, since a global view of all devices in an opportunistic network is not possible. Therefore, a device in an opportunistic network can only use either (i) its own context information or (ii) a partial view of the network through information shared by other devices via opportunistic contact. Based on this observation,

we devise three categories for handover mechanisms, i.e., *naive*, *work stealing*, and *local optimization*. A device using *naive* mechanism only requires its own resources utilization as context to make handover decision; while a device using *work stealing* and *local optimization* requires shared context from other devices. The details of each devised mechanisms will be elaborated in the following.

A. Naive

Naive mechanism does not require any sophisticated shared context; the decision is made by single device's context with respect to the resource utilization on this device. To this end, each participating device in our system maintains a queue of ATMT tasks. The size of ATMT tasks queue indicates the total resource, which a device can contribute. Two options are possible for naive handover. (i) Since an ATMT message represents a complex task that requires the execution of several services in a predefined order, the successful completion of a task is not guaranteed in opportunistic network. Consequently, to increase the success rate, a naive node simply contributes all of its resource available in ATMT tasks queue and passes the processed ATMT tasks to all neighbours. This behavior resembles the well-known *epidemic* routing [18]. Hence, the common observed characteristics of *epidemic* routing can also be applied for our naive mechanism; i.e., the success rate is improved by scarifying communication and computation overhead. Due to this reason, a naive mechanism utilizing full resources of participating devices, serves well as the baseline for benchmarking purpose. (ii) It can also be observed that, in dense opportunistic networks, a high number of devices providing similar services can exist. On the one hand, the resource on these devices will be used redundantly, following a greedy naive behavior. On the other hand, the success rate when reducing the size of ATMT tasks queue and rejecting ATMT tasks upon reaching a limit, can be compensated by the high number of participating devices with similar capabilities. In such cases, reducing the size of the ATMT tasks queue and rejecting tasks can decrease the number of redundantly executed operations, while preserving the high success rate and leading to improved load balancing. This intuition will be analyzed later in the evaluation (cf. Section VI). In summary, a naive device in our system will either fully utilize all its available resource, i.e., epidemic flooding of ATMT tasks in the whole network, or a device can intentionally reduce its tasks queue and drop upcoming received tasks.

B. Work Stealing

The term *work stealing* is coined in the context of parallel computing [19]; it refers to the act of an underutilized processor *stealing* threads from over-utilized processor, aiming to relieve over-utilized processors from high workload, thus a better load balancing among processors can be achieved. Fernando et al. [10] incorporates the concept of *work stealing* in the context of *mobile crowd computing*. Our devised *work stealing* strategy extends this idea for a more decentralized dynamic system, i.e., mobile devices in opportunistic network with the ability to act autonomously.

In our system, each operator device is qualified as a work stealer, i.e., if an operator device deems itself to be under-utilized, this device can ask to take over ATMT tasks from the nearby devices. Underutilization is determined based on the current number of ATMT tasks in the tasks queue. If this number is less than a work stealing limit, then an operator device will ask the surrounding operators to handover ATMT tasks. An operator device triggers the work stealing process by sending a work stealing message, indicating the number of ATMT tasks (n_{ws}) that this work stealing operator is willing to accept and the list of its providing operations. In order not to exhaust the maximum resource of the work stealing operator, n_{ws} should not exceed the maximum size of the ATMT tasks queue on the device. Furthermore, to avoid egoistic behavior of the participating operators, when receiving the work stealing message with the indicated capacity n_{ws}, a device is allowed to handover maximum up to n_{ws} tasks, however a minimum number of task n_{keep} should always be kept back in the tasks queue. To decide how many tasks should be handed over to the work stealing operator, three options are conceived: (i) Devices receiving work stealing message try to exploit the maximum capacity indicating by the work stealing operator without any coordination from the work stealing device. (ii) The work stealing device assumes the local coordination and divides the number of allowed ATMT tasks equally for its neighbors. (iii) The work stealing device accepts tasks from its neighbors following *first come first serve* principle. As soon as the maximum threshold is reached, the work stealing device will notify the neighboring devices to stop handing over tasks.

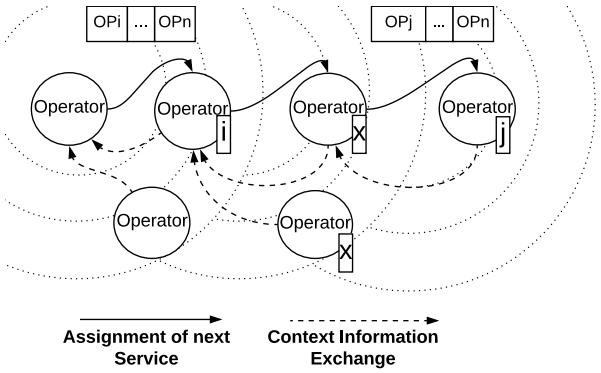


Fig. 3: Illustration of local optimization concept, choosing to the best next handover destination benefiting load balancing.

C. Local Optimization

Local optimization is inspired by the observation of Eager et al. [20], that a simple load adaptation locally in a distributed environment can lead to the overall improved performance of the whole system. Additionally, in the context of services composition in opportunistic network, Sadid et al. [13] show that the overall performance of opportunistic hop by hop composition is comparable to the performance of composition

orchestrated by a centralized entity. Following this line of thought, we devise strategies for tasks handover decision at an operator device in our system, requiring only local knowledge obtained through shared context of the neighboring devices. Our target is to optimize the load sharing among several devices locally by handing over the tasks to the next best destination within a close proximity. The local optimization is done by single devices autonomously, but still in a collaborative manner through shared context. The overall workflow of *local optimization* strategy is illustrated in Figure 3.

In opportunistic networks, the context of devices can be shared either in a reactive or proactive manner. Reactive context sharing is triggered only if a device receives an explicit query asking for its context. However, in a highly dynamic environment, a long time might elapse since the query is sent, until the information comes back to the query initiator. Due to this reason, proactive context sharing seems to be more favorable in opportunistic network. The context information is thus exchanged at any opportunistic contact of two devices in our system. Two devices exchange the summary of the context information about themselves and about the other devices that these two have seen in the past. Through this way, every devices have a snapshot of the shared context information. The context information required for local optimization of load balancing are generated by each device as a list of available operations ($op_i..op_j$), the currently-used capacity (n_u), the current position ($(long, lat)$), moving direction ($\vec{v}$) and a time stamp (t_{info}) when generating context information. When an operator device triggers the local optimization, it checks the current shared context and filters the nodes within a proximity of distance d_{max}, that possess the required operations, as potential destinations for task handover. The potential destinations can be further filtered, omitting devices that have distance around d_{max} and currently move farther away from the initiating device. To choose destinations benefiting the load balancing, we use a cost function covering three aspects for local optimized assignments, which are the currently-used capacity in the tasks queue (n_u), the distance and the uncertainty of the shared context information about operator O, i.e., ($\mu(N_O)$). The cost function is defined as follows:

$$c(N_A, N_O, \#_{OP}) = (w_l * c_l * \#_{OP} + w_d * c_d) * \mu(N_O) \quad (1)$$

in which:

$$\begin{aligned} \mu(N_O) &= 1 + \frac{t_{current} - t_{info}}{t_{keepAlive}} \\ c_l(N_O) &= \frac{n_{max} - n_u}{n_{max}} \\ c_d(N_A, N_O) &= \frac{d(N_A, N_O)}{d_{max}} \end{aligned} \quad (2)$$

In Equation 1, N_A is the node that wants to trigger the handover, to assign some of its tasks to other operator; N_O is a potential destination operator, to which the tasks can be assigned. $\#_{OP}$ is the number of operations that will be handed over. w_l and w_d are weighting factors for cost values of load (c_l) and distance (c_d), respectively. In Equation 2, the uncertainty factor $\mu(N_O)$ is captured using the time elapsed since the context information of operator O are generated until recently. The main cause of the uncertainty is the high dynamic of the network, caused by mobility or by disappearance upon exhaustive utilization of the devices. Consequently, outdated context information, which results in a higher uncertainty factor $\mu(N_O)$, can lead to a negative handover decision, increasing the total cost. The cost for load component in the equation is considered based on the number of currently utilized tasks in the tasks queue and the maximum size of the task queue (n_{max}). The distance component is determined by the ratio between the current distance $d(N_A, N_O)$ from the assigner to the operator and the search radius (d_{max}), as in Equation 2. This is based on the intuition, that the communication overhead for a nearer node is less than that for the farther node; since more hops might be required to reach an operator at larger distance.

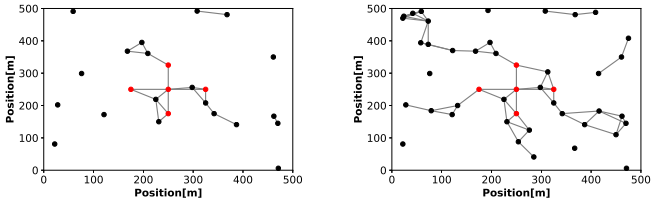
In order to improve load balancing, each device can trigger the local optimization to find the best destination with minimum handover cost for the upcoming operations of an ATMT task. We propose two modes to trigger local optimization to find the best next handover destination, i.e., (i) proactive mode: every time the shared context information are updated, indicating possible better destination for the next handover or (ii) reactive mode: only when a device receives more tasks than the current size of its task queue, indicating over-utilization. Regardless of trigger modes, to ensure effective dissemination of shared context information, every time a device detects a new neighbor, this device exchanges its summarized context information with the new neighbor.

VI. Evaluation

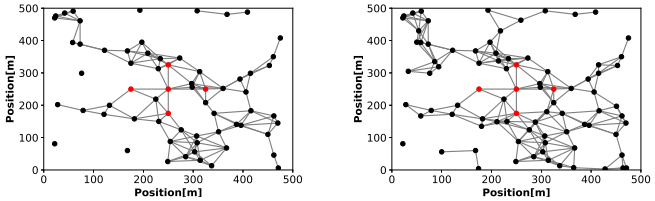
We implement and evaluate the task handover mechanisms as detailed in Section V, using a customized OMNeT++ module compatible with our designed ATMT message [5]. In this section, we first elaborate on the evaluation methodology, the simulation setup, and the evaluation metrics. Next, we study each handover mechanisms independently w.r.t. the evaluation metrics to identify the best performing option within each category. Last, we compare the proposed handover mechanisms against each other and point out the trade-off between the performance and load-balancing metric.

A. Scenario Modelling, Setup and Evaluation Metrics

Since the main target of our evaluation is the analysis of computation balancing, we model a simulation scenario to enable the dissemination of ATMT tasks into an opportunistic network. This network consists of several mobile nodes that move around a $500 \times 500m^2$ simulation area. Two nodes can communicate within $75m$ WiFi range. We abstract from a WiFi ad hoc model to enhance the scalability of the simulation and assume that the congestion will be handled by Link Layer mechanisms [21]. We set up five static nodes, one main delegator and four helper delegators which are connected to the main delegator. The main delegator generates ATMT-tasks and

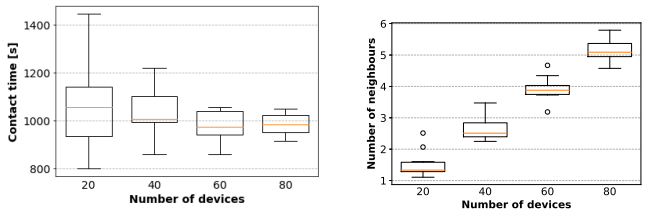


(a) 20 devices (b) 40 devices



(c) 60 devices (d) 80 devices

Fig. 4: Contacts among nodes for varied number of devices.



(a) Contact duration (b) Number of neighbors

Fig. 5: Average contact duration and number of neighboring devices according to the used Levy Walk mobility model.

injects these tasks to the network through the helper delegators. The reason for this particular setup is to allow the initial ATMT tasks to reach more operator nodes even under sparse network as in case of 20 devices (cf. Fig. 4a), aiming solely at generating a similar start configuration in both dense and sparse setups. The performance of the handover mechanisms, which rely on the behavior of the participating nodes during the simulation run, is not affected by this setup. We create two types of task; a simple task which contains between two or three operations, and a complex task which always contains five operations. The delegator nodes are marked in red as shown in Fig. 4. To control the movement of the simulated mobile nodes, we use the Levy Walk mobility model. This decision is based on the fact, that the Levy Walk mobility model is reported in [22] to resemble the human mobility patterns. We generate mobility traces accordingly using BonnMotion [23]. The direct contacts among mobile nodes from the generated traces are illustrated in Fig. 4. Fig. 5 shows the observed characteristics of the generated traces, which suggest a longer, more stable contact duration and an increasing number of direct neighbors with more devices in the network. As such, 20 nodes represent a sparse opportunistic network, while 80 nodes represent a dense opportunistic network. The most important simulation parameters are summarized in Table I.

TABLE I: Simulation Setup

Simulated Area Size	$500 \times 500\, m^2$
Simulation Time	one hour
Number of Nodes	20, 40, 60, 80
WiFi Transmission Range	$75\ m$
Mobility Model	`LevyWalkMobilityModel`
#ATMT-Tasks	100, 1000
Naïve Greedy	full, limited
Work Stealing	full, FCFS, equalized
Local Optimization	proactive, reactive

We repeated each simulation ten times and plotted all obtained results with 90% confidence intervals. The following evaluation metrics were used to analyze the results:

(a) *Success rate* denotes the ratio between the number of successfully completed ATMT tasks that can be delivered back to the main delegator and the total number of tasks.

(b) *Communication overhead* is defined as the total number of ATMT messages that are generated and duplicated by the handover strategies.

(c) *Completion time* is the time elapsed since the main delegator injects tasks into the network, until all processed results come back to the main delegator.

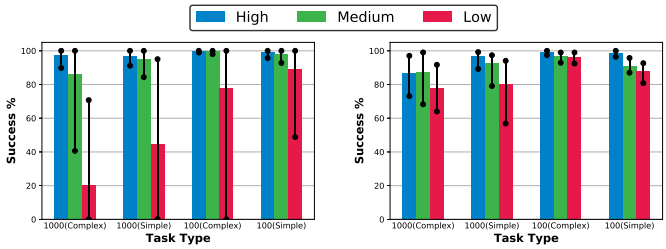
(d) *Jain index* is proposed by Jain et al. in [24] as follows: $JI(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n * \sum_{i=1}^{n} x_i^2}$, wheres x_i denotes the resource consumed (in our scenario the number of operations executed) by node i. Jain index with value closer to 1 indicates higher fairness among the resources consumed by all nodes. Thus, Jain index is able to quantify the quality of load balancing mechanisms.

(e) *Redundancy factor* is defined as the ratio between the number of redundantly executed operations and the original number of operations in the network.

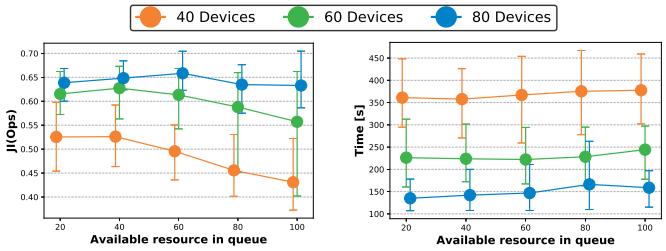
B. Handover Mechanisms Analysis

Naive: The evaluation for naive mechanisms has two objectives: (i) assessment of ATMT tasks dissemination in scarce and dense opportunistic networks and (ii) identification of suitable tasks queue's size which benefits load balancing quality as a baseline for further analysis.

In our simulation, each node possesses a number of predefined services which this node can execute. For evaluation of naive mechanisms, we set up three different classes characterizing the availability of the services on all nodes, i.e., *high, medium, low*. The distribution of the services availability on the nodes in each class follows a normal distribution. The *high* class assigns 50% of the nodes with all 5 available services required for the operations defined in the ATMT task; the *medium* class assign 50% of the nodes with between 2 and 3 available services; and the *low* class assign 50% of the nodes with no services, the majority of the rest are assigned only 1 single service. Fig. 6a and 6b show the dependency of success rate on the availability of the services. Low services availability decreases the success rate, which is visible in case complex tasks are executed in sparse network with only 20

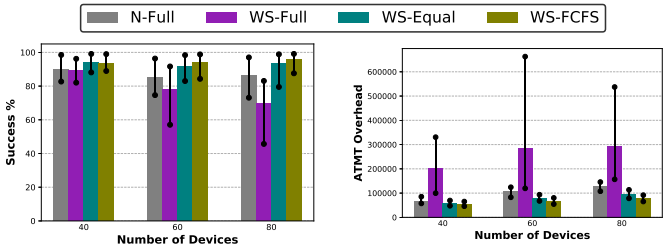


(a) Success rate with 20 nodes (b) Success rate with 80 nodes

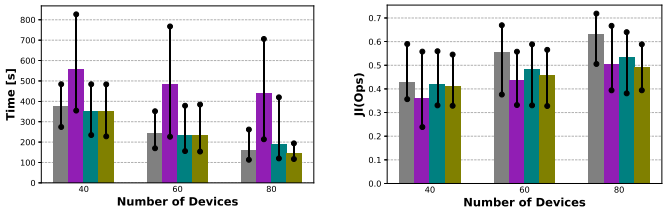


(c) Jain index (d) Completion time

Fig. 6: Analysis of naive handover.

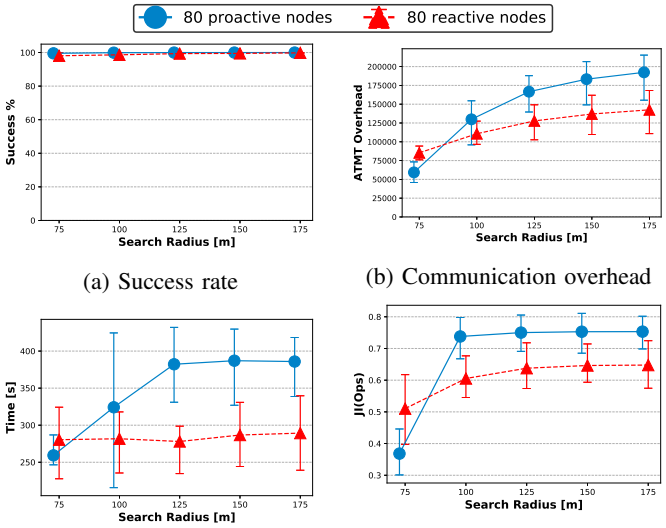


(a) Success rate (b) Communication overhead



(c) Completion time (d) Jain index

Fig. 7: Analysis of work stealing mechanisms.



(a) Success rate (b) Communication overhead

(c) Completion time (d) Jain index over executed operations

Fig. 8: Analysis of different modes for local optimization.

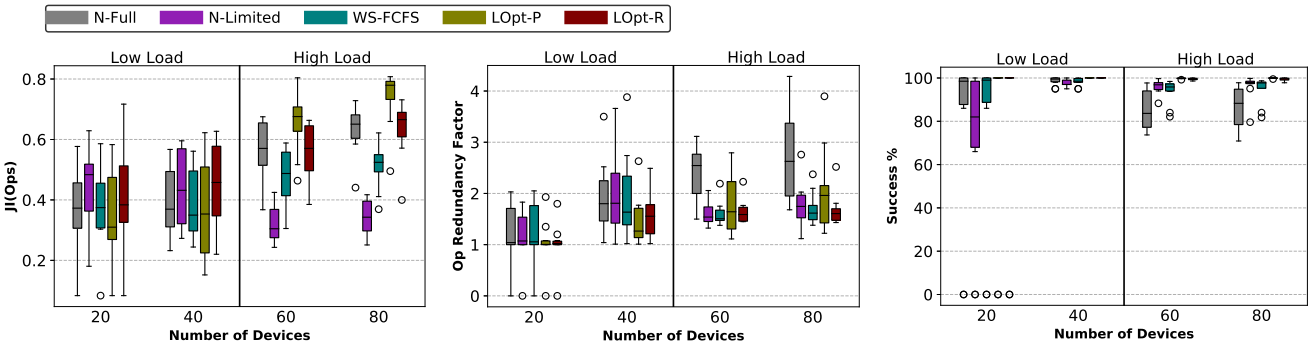
nodes. The success rate in this case only reaches around 20% (cf. Fig. 6a). However, the success rate despite low services availability can be compensated through higher number of devices as we anticipated. Fig 6b shows, that the success rate for complex tasks with low services availability can be improved from 20% (with 20 nodes) to 80% (with 80 nodes).

The effect of tasks queue's size on the performance and the quality of load balancing was examined. Fig. 6c shows slightly better values for Jain index over the executed operations when decreasing task queue's size, compared to the maximum size (100 in our simulation), indicating slightly improved fairness in the system. Shorter queue size also means less resource has to be contributed by the nodes. With respect to the completion time, a shorter tasks queue does not have any negative effect. Rather, the completion time depends on the number of devices, i.e., faster completion time can be achieved with more devices in the network as shown in Fig. 6d. Overall, the analysis of naive handover mechanisms suggests reducing the size of the tasks queue, thus frees resources for participating nodes.

Work Stealing: We compare thee options for work stealing as introduced in Section V-B against naive flooding handover mechanisms (*N-Full*). The 3 options for work stealing are respectively: *WS-Full* which tries to exploit the full capacity of the work stealing node, *WF-Equal* in which the work stealing node divides the accepted capacity equally among neighbors, and *WS-FCFS* which follows first come first serve principle.

It can be observed that greedy behavior when handing over tasks in *WS-Full* decreases the success rate (down to 70% with 80 nodes), while generating even more communication overhead compared to the naive handover *N-Full*. This negative effect is due to the redundant task handovers triggered by the neighbors in *WS-Full*, which the work stealing nodes have to drop at overloaded capacity. On the contrary, the two other work stealing options, *WS-Equal* and *WS-FCFS* slightly improve the success rate and even the completion time in some cases compared to *N-Full* (cf. Fig. 7a, 7c). The reason is, work stealing with *WS-Equal* and *WS-FCFS* can free some resources of the nodes locally; in contrast, naive handover mechanism generates more redundant operations (cf. overall comparisons, Fig. 9b). However, depending on the distribution of the nodes in the area, the chance for a work stealing node and an overloaded node to meet cannot always be guaranteed. Correspondingly, Jain index values obtained through *work stealing* display no major load balancing improvement using *work stealing* concept (cf. Fig. 7d).

Local Optimization: Since the local optimization looks for the best next destination within a search radius to assign the handover, we anticipate the size of this search radius affects



(a) Jain index over executed operations. (b) Redundancy factor. (c) Success rate.

Fig. 9: Comparison of handover mechanisms.N-Full denotes the flooding based naive handover; N-Limited denotes the naive handover with limited task queue; WS-FCFS represents work stealing, using first come first serve; LOpt-P denotes the *proactive local optimization*; LOpt-R denotes the *reactive local optimization.*

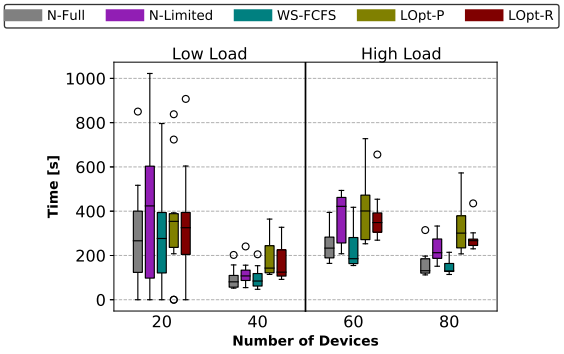


Fig. 10: Completion time of handover mechanisms.

all evaluation metrics. Hence, we vary the size of the search radius and analyze the corresponding influences. The results are shown in Fig.8. We evaluate two modes of the local optimization as introduced in Section V-C, i.e., proactive mode which triggers the local optimized handover upon receiving new shared context, and reactive which triggers the local optimized handover only for overloaded situations.

The success rate for both proactive and reactive modes are high (almost always at 100%) regardless of the size of the search radius. Obviously, larger search radius leads to more communication overhead. *Proactive local optimization* generates more communication overhead compared to *reactive local optimization*; since the context in an opportunistic mobile network tends to change rapidly, leading to more frequent information exchange in proactive mode (cf. Fig. 8b). A longer completion time for *proactive mode* is visible when increasing the size of the search radius, which is the trade-off for obtaining better result for optimization. In contrast, the completion time for *reactive mode* is quite stable, since it only triggers the local optimization at circumstances (cf. Fig. 8c). Fig. 8d shows improved Jain index values with larger search radius. With *proactive mode*, the Jain index value increases from 0.37 with 75 m search radius, up to 0.75 with 125 m search radius. *Reactive mode* increases the Jain index value from 0.5 at 75m, up to 0.65 at 125 m. Increasing the search radius more than 125 m shows no more fairness improvement, suggesting converge quality for load balancing. Hence, the search radius should be restricted in order not to waste communication overhead. Between two modes, *proactive local optimization* yields better quality for load balancing than *reactive mode* at larger search radius. This can be explained by the fact, that *proactive mode* reacts on the context changes of the network, while *reactive mode* waits for an overloaded situation.

C. Handover Mechanisms Comparison

Having analyzed the handover mechanisms individually in Section VI-B, we now compare all mechanisms against each other. To cover the performance indicators for both sparse and dense network situations, we use two setups: (i) a *low load* setup with 100 tasks distributed to 20 or 40 nodes and (ii) a *high load* setup with 1000 tasks distributed to 60 or 80 nodes. Selected results for the comparison regarding the Jain index, redundancy factor, success rate and completion time are presented accordingly in Fig. 9a, 9b, 9c, 10. For a sparse network, the quality for load balancing fluctuates, regardless of handover mechanisms. It is to be expected, since a sparse opportunistic network tends to be partitioned; many nodes are therefore isolated the whole time, providing no way for their resources to be exploited. Evidently, the quality for load balancing can be improved with more nodes in the network. Fig. 9a shows that our proposed *proactive local optimization* can achieve the best Jain index value (around 0.8 in case of 80 nodes), outperforms other handover mechanisms. The quality of load balancing obtained by *reactive local optimization*, despite being less than *proactive local optimization*, is still comparable to flooding based naive handover (both achieve Jain index values at around 0.65 with 80 nodes). *Proactive local optimization* yields the lowest redundancy factor (at avg. 1.5), compared to a very high redundancy factor of N-Full (at avg. 2.5, the worst case up to more than 4) (cf. Fig. 9b).

This confirms that the resources in naive mechanisms are used redundantly, while our proposed local optimization mechanisms help to alleviate this problem. As already discussed in the analysis of *work stealing*, *work stealing* cannot improve the overall load balancing, but can achieve higher success rate compared to naive mechanisms. The result shown in Fig. 9c again confirms this observation. The same result also demonstrates that our proposed local optimization mechanisms not only outperform other mechanisms w.r.t. load balancing, but are also able to outperform others w.r.t. success rate. Moreover, the marginal variances shown in the box plot obtained from the results of both local optimization modes, prove the robustness of the mechanisms, against the rapid changes in dynamic, mobile networks. The improvements achieved by local optimization mechanisms, however, have to take into account longer completion time (cf. Fig. 10).

VII. Conclusion and Future Work

In this paper, we extended the adaptive task-oriented message template (ATMT) defined in our previous work [5] and proposed several handover mechanisms that enable load balancing for distributed processing of complex tasks. Our proposed mechanisms were designed focused mainly on opportunistic networks, thus do not require any centralized coordination. The evaluation results show that we were able to achieve better load balancing through local optimization, leveraging only locally shared context information. Overall, our proposed task message template facilitates distributed coordination and is thus suitable for decentralized, highly dynamic environment.

Several directions are possible as our future work. First, the load balancing mechanisms proposed in this work can be further evaluated using real hardwares, which allows us to determine over-utilized situation in realistic conditions, e.g., based on CPU load or energy consumption level. This will also allow us to incorporate, and consequently study the effect of heterogeneity in terms of hardware configuration, energy consumption when executing a complex operation on distributed load balancing. Second, the handover mechanisms, especially work-stealing can be further augmented by prioritizing tasks, i.e., setting higher handover priority for nearly completed tasks can benefit the success rate, while setting higher priority for computation-intensive tasks will work in favor of load balancing. Third, within the context of information centric ad hoc network (ICN), it is shown that situational data can be collected by mobile devices [14]. Hereby, we want to combine the design of ATMT with data transport phase in ICN to deliver processed high-valuable information to the query initiator.

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the NICER project and by the German Federal Ministry of Education and Research (BMBF) Software Campus project "OppEPM" [01IS12054].

References

[1] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, and J. Crowcroft, "A Survey of Opportunistic Offloading," *IEEE Communications Surveys & Tutorials*, 2018.

[2] C. Meurisch, J. Gedeon, T. A. B. Nguyen, F. Kaup, and M. Mühlhäuser, "Decision Support for Computational Offloading by Probing Unknown Services," in *IEEE ICCCN*, 2017.

[3] E. Borgia, R. Bruno, M. Conti, D. Mascitti, and A. Passarella, "Mobile edge clouds for Information-Centric IoT services," in *IEEE ISCC*, 2016.

[4] C.-K. Tham and R. Chattopadhyay, "A Load balancing Scheme for Sensing and Analytics on a Mobile Edge Computing Network," in *IEEE WoWMoM*, 2017.

[5] T. A. B. Nguyen, C. Meurisch, S. Niemczyk, D. Böhnstedt, K. Geihs, M. Mühlhäuser, and R. Steinmetz, "Adaptive Task-Oriented Message Template for In-Network Processing," in *IEEE NetSys*, 2017.

[6] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *SIMUtools*, 2008.

[7] B. J. Bonfils and P. Bonnet, "Adaptive and Decentralized Operator Placement for In-network Query Processing," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 389–409, 2004.

[8] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, and U. Ramachandran, "Mcep: A mobility-aware Complex Event Processing System," *ACM Transactions on Internet Technology*, vol. 14, no. 1, p. 6, 2014.

[9] F. Fossati, S. Moretti, and S. Secci, "A Mood Value for Fair Resource Allocations," in *IFIP Networking*, 2017.

[10] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile Crowd Computing with Work Stealing," in *IEEE NBiS*, 2012.

[11] A. Benchi, P. Launay, and F. Guidec, "Solving Consensus in Opportunistic Networks," in *ACM ICDCN*, 2015.

[12] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili, "Uncertainty-aware Autonomic Resource Provisioning for Mobile Cloud Computing," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 8, pp. 2363–2372, 2015.

[13] U. Sadiq, M. Kumar, A. Passarella, and M. Conti, "Service Composition in Opportunistic Networks: A Load and Mobility aware Solution," *IEEE Transactions on Computers*, vol. 64, 2015.

[14] T. A. B. Nguyen, P. Agnihotri, C. Meurisch, M. Luthra, R. Dwarakanath, J. Blendin, D. Bohnstedt, M. Zink, R. Steinmetz *et al.*, "Efficient Crowd Sensing Task Distribution Through Context-aware NDN-based Geocast," in *IEEE LCN*, 2017.

[15] P. Lampe, L. Baumgärtner, R. Steinmetz, and B. Freisleben, "Smartface: Efficient face detection on smartphones for wireless on-demand emergency networks," in *IEEE ICT*, 2017.

[16] R. Dwarakanath, B. Koldehofe, Y. Bharadwaj, T. A. B. Nguyen, D. Eyers, and R. Steinmetz, "TrustCEP: Adopting a Trust-Based Approach for Distributed Complex Event Processing," in *IEEE MDM*, 2017.

[17] A. M. Alakeel, "A Guide to Dynamic Load Balancing in Distributed Computer Systems," *International Journal of Computer Science and Information Security*, vol. 10, no. 6, pp. 153–160, 2010.

[18] A. Vahdat and D. Becker, "Epidemic Routing for partially connected Ad Hoc Networks," Duke University, Tech. Rep., 2000.

[19] R. D. Blumofe and C. E. Leiserson, "Scheduling Multithreaded Computations by Work Stealing," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 720–748, 1999.

[20] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in homogeneous Distributed Systems," *IEEE transactions on software engineering*, no. 5, pp. 662–675, 1986.

[21] C. Lochert, B. Scheuermann, and M. Mauve, "A Survey on Congestion Control for Mobile Ad Hoc Networks," *Wireless communications and mobile computing*, vol. 7, no. 5, pp. 655–676, 2007.

[22] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the Levy-Walk Nature of Human Mobility," *IEEE/ACM transactions on networking (TON)*, vol. 19, no. 3, pp. 630–643, 2011.

[23] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, "BonnMotion: A Mobility Scenario Generation and Analysis Tool," in *SIMUTools*, 2010.

[24] P. N. D. Bukh and R. Jain, "The Art of Computer Systems Performance Analysis, Techniques for experimental Design, Measurement, Simulation and Modeling," 1992.

PPAD: Privacy Preserving Group-Based ADvertising in Online Social Networks

Sanaz Taheri Boshrooyeh
Koç University, İstanbul, Turkey
staheri14@ku.edu.tr

Alptekin Küpçü
Koç University, İstanbul, Turkey
akupcu@ku.edu.tr

Öznur Özkasap
Koç University, İstanbul, Turkey
oozkasap@ku.edu.tr

***Abstract*—Services provided as free by Online Social Networks (OSN) come with privacy concerns. Users' information kept by OSN providers are vulnerable to the risk of being sold to the advertising firms. To protect user privacy, existing proposals utilize data encryption, which prevents the providers from monetizing users' information. Therefore, the providers would not be financially motivated to establish secure OSN designs based on users' data encryption. Addressing these problems, we propose the first Privacy Preserving Group-Based Advertising (PPAD) system that gives monetizing ability for the OSN providers. PPAD performs profile and advertisement matching without requiring the users or advertisers to be online, and is shown to be secure in the presence of honest but curious servers that are allowed to create fake users or advertisers. We also present advertisement accuracy metrics under various system parameters providing a range of security-accuracy trade-offs.**

I. Introduction

Online Social Networks (OSN) such as Facebook, Twitter, and Google+ are in the center of people's attention due to the functionality and networking opportunities they offer. OSNs consist of three main entities: *Server, user*, and *advertiser*. *Server* supports the OSN's functions using its storage and computational resources. Users are able to share their personal information with each other and establish new friendships via OSN. Advertisers ask *Server*'s help to detect their target customers out of OSN's users. Despite the attractive services that OSNs offer to the users, sharing personal information with these networks raises privacy problems where servers monetize users' information by selling them to the advertising companies [24], [27]. To prohibit the accessibility of OSN providers to the plain information of users, secure OSN designs that employ data encryption are proposed [12], [11], [31], [33], [3], [4]. While these solutions provide tangible benefits to the users' privacy, they neglect the role of advertiser as part of the OSN, which results in monetizing *inability* for the server [32]. Thus, OSN servers are left with no financial motivation to establish such secure OSN services.

The lack of a convincing commercial model for secure OSNs is our main motivation to propose a Privacy Preserving ADvertising (PPAD) system for OSNs. PPAD can be incorporated into secure OSN designs (where the OSN's functionality meet data confidentiality) to provide advertising service. Yet, achieving the best of both worlds is impossible: we provide a trade-off between personalized advertising accuracy and user profile privacy. We first define these terms, explain why it is impossible to achieve some goals simultaneously, and show how we achieve a solution whose parameters can be tweaked for various settings.

ISBN 978-3-903176-08-9

In PPAD, we introduce the notion of **group-based advertising** on the encrypted data. By group-based advertising, we aim to cope with the security issues raised by the personalized counterparts [36], [21]. In fact, performing personalized advertising on the encrypted data will ultimately violate user privacy. The reason is that knowing that a particular advertising request (which is a set of attributes) is matched to an encrypted profile implies that the profile entails the attributes listed in that request. Therefore, although the matching is performed on the encrypted data, the server is able to learn the profile content i.e., user's attributes. This cannot be prevented using *any* (cryptographic) method unless one (unrealistically) assumes that the adversarial server cannot create fake advertisement requests targeting known attributes.

One remedy of this problem is that the final matching result must be computed in the encrypted format (server does not learn the result) and then the results are sent to the user to open and read. However, this approach is cumbersome as the user has to open (decrypt) all the matching results (which is linear in the total number of advertising requests) and retrieve the matched advertisements from the server in an oblivious way (which again incurs a high load).

Due to this privacy concern, we define a new advertising paradigm called group-based advertising. In short, we (randomly) partition users into groups of equal size at the registration phase. Then, each advertising request is compared to the profiles of a group of users and not a single user. The matching result indicates whether there exist some threshold-many target users among the group members. If it happens, then the advertising is shown to *all* the group members. Note that the matching result reveals neither the identity of the matched user nor the number of matched users but only the existence of at least threshold-many matches. By this method, the matching result is unlikable to an individual profile. We propose a formal security definition to capture this notion of unlinkability.

Another property of PPAD is to keep the advertising procedure **transparent** to the users/advertisers, similar to the insecure counterparts. That is, users and advertisers carry no overhead except uploading their data to the social network. Henceforth, the matching process is operated only by the

server and needless to any constant online connection of the user or the advertiser. Prior solutions [34], [15], [6], [17], [29], [22] fail to provide the transparency feature. User's involvement in the matching procedure adds an overhead to the user that grows linearly with the number of advertising requests. This overhead demotivates the user from participating in the PPAD protocol as the user is obliged to stay online until the server matches user's profile to the advertising requests. It also negatively affects the system's efficiency as the servers' working-time depends on the users' online time. In PPAD, users receive relevant advertisements, which are found by the server during the users' and advertisers' off-time. We enable this by utilizing a **privacy service provider** (PSP) that assists OSN providers to protect the privacy of their users. A PSP can be a non-profit or governmental entity that can help users and multiple providers achieve privacy-preserving advertising and can be implemented with low cost. Due to their fame and reputation, PSPs are assumed to be non-colluding parties and hence are used in similar privacy-concerned applications [35].

Our contributions in this paper are as follows.

- We propose PPAD advertising system that preserves user privacy and is applicable on the secure OSNs where users' information are encrypted.
- In contrast to the existing solutions where secure matching requires both user and advertiser to be permanently or simultaneously online, PPAD allows users and advertisers to be *offline* after the registration. Once the matching is performed offline by the server, the advertisement is shown to relevant users the next time they appear online (or via push notifications, etc.).
- We present a formal security definition for user privacy. We argue that a meaningful security definition in this setting must allow the adversarial server to control some advertisers and users. Our system is formally proven to be secure against the privacy service provider as well as the server that may additionally employ a number of fake users or advertisers.
- We define two performance metrics of *Target accuracy* and *Non-Target accuracy* to be used in group-based advertising systems. Using empirical analysis, we capture the effect of group size and threshold value on the system performance and discuss their security implications.

II. System Overview

Model: An overview of PPAD is shown in Figure 1. The participants are users, advertisers, and the OSN provider (*Server*) who gets help from a third party that is a privacy service provider (*PSP*). In PPAD, the advertiser specifies the attributes of its target users, and uploads an advertising request to the *Server* as a Bloom filter. We define an advertising request to be a conjunction of several attributes e.g., $\{Artist, Player, Scientist\}$. Users are (randomly) partitioned into groups of size k at the registration phase. As discussed, this grouping is *necessary* for user privacy, and must not be done based on similar interests. To preserve confidentiality, users **encrypt** their Bloom filters using additive homomorphic encryption and secret sharing techniques before submission to

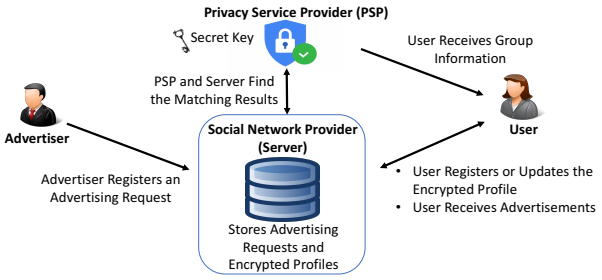


Fig. 1: PPAD system overview

Server. To provide provable security, PPAD requires users to encrypt their data using two different public keys PK_1 and PK_2. The decryption power i.e. the corresponding secret keys SK_1 and SK_2 are given only to *PSP*, but *PSP* never receives these individual profiles.

For each group, if at least threshold-many group members' profiles entail the same attributes listed in the advertising request, the group is marked as a target group. The advertisement is then presented to *all* members of the target groups (since advertising to a subset would require violating privacy). Afterwards, the social network provider charges the advertiser based on the number of target groups (hence users) found for its advertising requests. The group size and threshold are parameters that affect both advertisement accuracy and user privacy. We analyze this trade-off in Section V.

Security Goals: In PPAD, our security goal is to protect the link-ability of a successful match to a specific member. That is, when a group is marked as a target group, the server should not be able to say which members of that group exhibit the attributes in the advertising request.

However, it is important to realize that there is always an implicit and inherent privacy leakage in any advertising system, regardless of how secure it is designed. This leakage is that as soon as the server obtains the matching result, it understands the inclusion or exclusion of some specific attributes among the group members, although the matching is performed entirely on the encrypted profiles. In group-based advertising, the inclusion or exclusion of attributes in the group is unlinkable to a particular group member, while in the personalized counterpart, the matching result immediately breaks user privacy.

Security Assumptions: What we presume in PPAD is that the main adversary of user privacy is the social network provider, i.e., *Server*, (or an attacker controlling it) who may be curious to link the matching results of each group to the individual members. In this regard, the server may employ polynomially-many advertisers and take control of some of the users in each group. If the *Server* manages to control all but one member of some group, the same arguments against the impossibility of providing privacy in a personalized advertisement system apply. Therefore, we necessarily assume that at least two users per group are honest.

Moreover, the *Server* is assumed not to be able to collude with the privacy service provider, i.e., *PSP*. We believe that such a non-profit or governmental organization would not cooperate with the social network provider against user privacy

due to the fame and reputation concerns. Hence, privacy service providers are assumed to be non-colluding parties and are used in similar privacy-concerned applications [35]. In Section VI-B, we formally prove that neither *PSP* nor the *Server* would be able to violate user privacy.

Since we employ *PSP* as an external entity, it is important that it can be implemented with low cost, requiring minimal change in the OSN system. In PPAD, users contact *PSP* only during registration to receive some anonymous, non-personalized group parameters. Advertisers never need to contact *PSP*. Moreover, when *PSP* helps *Server* during the matching process, it performs two decryptions and some arithmetic operations per matching (6 milliseconds per matching). Section V-B presents more details on performance.

Preventing Compound Group Matching Although group matching preserves user privacy, the social network provider may learn the identity of the target users by arbitrarily combining profiles of users from different groups and analyzing the changes in the matching results. To avoid this misbehavior, users of each group are given zero-sum secret shares by *PSP*. They embed their secret shares in their encrypted profiles. Decryption of individual profiles with different embedded secret shares results in garbage values. Thus, any attempt by *Server* toward grouping arbitrarily chosen users' profiles fails. The only way to cancel out the secret sharing is to aggregate the profiles of users of the same group, as then the secret shares' summation would be zero. Thus, *Server* has to aggregate profiles of each group separately and sends the aggregated data to *PSP*. Finally, *PSP* decrypts and de-aggregates the data to find the number of matching users in the group. We enable aggregation and de-aggregation of profiles using super increasing sets (more details are presented in sections III and IV-A4).

Profile Update Insecurity Performing profile update in secure group-based advertising systems comes with a serious privacy issue, whose solution hugely degrades system efficiency. Essentially, when a member of group modifies her profile (by adding or removing some attributes), *Server* can analyze the changes in the matching results of that group (against advertising requests) before and after user's profile update and realize which attributes the user has modified in her profile. Also, the group-mates are vulnerable to the same security risk. Due to this security problem, if a user wants to modify her profile, she has to join a new group (similarly her group-mates), and the old group is now disfunctional. This is regardless of the underlying (cryptographic) tools employed.

Prior studies with the profile update functionality are not applicable to the context of advertising in social networks since they assume that the user does not share its profile with the server [34] or they employ an IP Proxy server [13] so that users anonymously add new preferences to their profiles. The former contradicts with the advertising transparency and degrades the performance. The latter is not applicable to OSNs since users access the social network via a particular account and hence the server observes that the update operation is done under a particular account.

III. Preliminaries

Bloom Filters: Bloom filters [8] are used to represent sets, and efficiently check whether an element belongs to a set. A Bloom filter is constructed with an array of p bits, initially zero, and d hash functions, $H_1(.)$,...,$H_d(.)$. p is called the size of the Bloom filter. To insert an element x_1 into the Bloom filter, all the hash functions are applied on x_1 (i.e. $i_1 = H_1(x_1)$,..., $i_d = H_d(x_1)$) and the array cells at indices corresponding to the hash outputs are set to 1. Testing the membership of an element is done by applying all the hash functions on it (similar to the insertion), and checking that whether all the corresponding indices are equal to 1. If one of them is not equal to 1, then that element does not belong to the set. Otherwise, with the false positive probability of $1-(1-((1-\frac{1}{p})^{d})^{e})^{d}$ the element belongs to the set, where e is the number of elements inserted into the Bloom filter. In the rest of the paper, $BFCreate(inSet)$ creates a Bloom filter with *inSet* being the set of input elements.

Super Increasing Set: A super increasing set [9] of length g is a series of g positive real numbers, $\{s_1, s_2, ..., s_g\}$, where each element is greater than the sum of its preceding elements i.e., $(\forall j \in \{2,...,g\} : s_j > \sum_{i=1}^{j-1} s_i)$.

Additive Homomorphic encryption scheme: A public key encryption scheme $(KeyGen, Enc, Dec)$ is called additive homomorphic [19] if for all m_0, m_1 from the message space $C_0 \odot C_1 = Enc(m_0) \odot Enc(m_1) = Enc(m_0 + m_1)$ where $\odot$ is an operation defined over ciphertexts. Example is Paillier encryption [28] where $\odot$ corresponds to the multiplication over ciphertexts.

Negligible Function: A function f is called negligible if $\forall$ positive polynomials $p(.)$ $\exists I$ *s.t.* $\forall i > I$ (where i is a real number): $f(i) < \frac{1}{p(i)}$.

Secret Sharing Secret sharing [5] is a method to disseminate a secret among a set of parties. Consider zero as the secret, one way to create k shares of zero is to generate $k-1$ shares randomly $(SS_i, i \in \{1,...,k-1\})$ and then set the last share to $SS_k = 0 - \sum_{i=1}^{k-1} SS_i$ mod q where q indicates the modulus. The length of the secret shares must be long enough to hide the data content (longer than the maximum data size). We define $SSGen(k,q)$ as a function which generates k zero-sum secret shares out of the given message space (i.e., modulus) q.

IV. PPAD

A. Full Construction

This section presents our full construction, explaining which party runs which algorithm at which stage. Throughout the explanations, $x \leftarrow X$ demonstrates picking an element x uniformly at random from set X, and $||$ represents concatenation.

The **OSN initialization** is launched by *PSP* to generate system parameters. Users register their encrypted profiles in **User Registration**. A profile is a modified variant of a Bloom filter whose elements are separately encrypted under an additive homomorphic encryption scheme. Advertisers engage in **Advertiser Registration** protocol to submit an advertising request as a Bloom filter. The *Server* cooperates with *PSP* in the **Advertisement** protocol to find the target groups for each advertising request. In short, for every group and advertising

request pair, the *Server* aggregates encrypted profiles of users in that group and sends the aggregate as well as the advertising request to *PSP*. Consequently, *PSP* checks if the group matches to the request and responds to *Server* accordingly.

1) OSN initialization (OSNInit)

Server: The *Server* initializes a database as *DB*.

PSP: *PSP* determines the security parameter 1^{λ} and the threshold value. It establishes an additive homomorphic encryption π=*(KeyGen,Enc,Dec)* scheme with message space *MSpace*, and generates (PK_1, SK_1) and (PK_2, SK_2) as two pairs of public and private keys. We need two sets of key pairs for the security proof to work. The reason will be apparent in Section VI-B.

2) User Registration (UReg)

UReg protocol is shown by Figure 2.

PSP: User connects to *PSP* via a secure and server authenticated channel to receive its group related information. Initially, *PSP* determines the group identifier *GID* of the user. *GID*s can be assigned according to the users' arrival i.e., the first k users are assigned to the first group and the second k users to the second group, etc. Note that a group needs to be full to be advertised to, since the secret shares will not sum up to zero otherwise. *PSP* generates a fresh set of k secret shares as $GSS = \{SS_1, ..., SS_k\}$ per group and assigns shares to the users.

Also, *PSP* assigns a delimiter, D, to each user such that D is unique among group-mates. Each user embeds its delimiter in the profile. The structure of delimiters is given in Equation 1. Delimiters exhibit the property of a superincreasing set and help in aggregation and de-aggregation of members' profiles during the advertisement protocol. *PSP* generates a set of k delimiters denoted by *DSet* once and uses them for every group.

$$\forall j \in \{1, ..., k\}, D_j > \sum_{i=1}^{j-1} D_i * p \tag{1}$$

User: To make a profile, users may enter their preferences into a well-structured form like a Facebook profile. However, the presentation of profile form to the users is an orthogonal issue to the PPAD. Ultimately, all the collected attributes (denoted by *AttSet*) are transformed to a modified version of a Bloom filter at the user side. In Algorithm 1, first a Bloom filter, Pf, is generated out of *AttSet*. Then, each bit i of Bloom filter i.e., Pf_i is updated to $Pf_i * D + SS$. In words, we replace the 0-bit values of Bloom filter with user's secret share and the set bit values with the summation of the user's secret share and delimiter. This modification helps in two regards, first, to enable aggregation and de-aggregation by the help of delimiters, and second, to prevent compound group matching using secret shares (see the advertisement protocol). Finally, each modified element of Bloom filter is encrypted under the public encryption key PK given as input to the Algorithm 1.

The user selects its username *UName*, and generates two encrypted profiles EPf and $E\hat{P}f$ by running Algorithm 1 under two public keys PK_1 and PK_2, respectively. Then it uploads its encrypted profiles EPf, $E\hat{P}f$ alongside *UName* and *GID* to *Server*.

Algorithm 1 PCreate(AttSet, D, SS, PK)

1: $Pf = BFCreate(AttSet)$
2: $EPf = \{Enc_{PK}(Pf_i * D + SS)\}_{1 \le i \le p}$
3: **return** EPf

Server: *Server* receives the encrypted profiles and inserts them into the database *DB*.

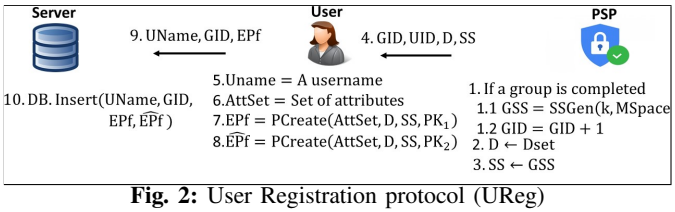


Fig. 2: User Registration protocol (UReg)

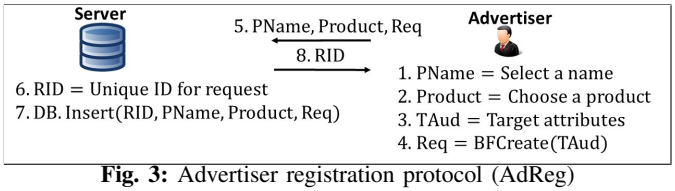


Fig. 3: Advertiser registration protocol (AdReg)

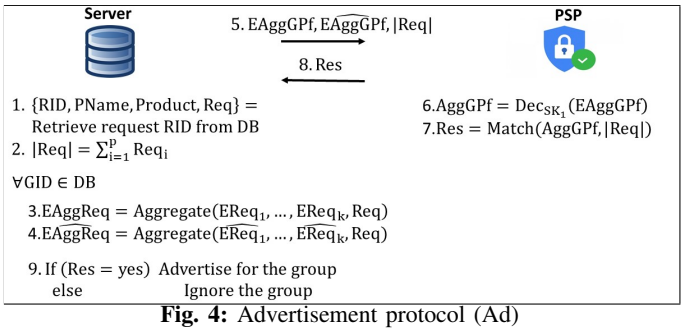


Fig. 4: Advertisement protocol (Ad)

3) Advertiser Registration (AdReg)

AdReg is shown in Figure 3. During *AdReg*, the advertiser registers its advertisement request under its name i.e. *PName* into the OSN. Advertiser specifies a set of attributes denoted by *TAud* for its target audience. The advertiser creates a request as a Bloom filter out of *TAud*. Then, it submits the Bloom filter, its name and the product to be advertised to *Server*. Advertising request preserves the AND operation between the targeted attributes. For example, a single request may target users with both attributes X **AND** Y. However, an advertising request which targets X **OR** Y must be split and submitted as two separate requests: one for X and the other for Y. *Server* registers the request, assigns a unique request identifier *RID*, and sends *RID* to the advertiser.

4) Advertisement (Ad)

Figure 4 represents the *Ad* protocol. During the *Ad* protocol, *Server* first retrieves an advertising request i.e., $(RID, PName, Product, Req)$ from *DB*. Next, to find the matching between the request and each group of users, *Server* and *PSP* interact as follows (**both users and advertisers are offline during this procedure**, which is one of our main contributions):

Server **Aggregate**: *Server* checks whether the advertising request is already matched against the group or not. If it is not matched, then *Server* retrieves profiles of group members and proceeds to the aggregation phase. As we already mentioned,

the aggregation helps cancel out the secret shares embedded in users' profiles (as discussed in Section II, the purpose of secret shares is to prevent compound group matching).

To aggregate profiles, we utilize the fact that a profile Pf (which is a Bloom filter according to Algorithm 1) matches the advertising request Req if for every set bit position of Req the corresponding bit in Pf equals to 1. More formally, Pf is a target for Req if

$$\forall 1 \leq i \leq p \quad s.t. \quad Req_i = 1 \quad \rightarrow \quad Pf_i = 1 \tag{2}$$

Stated differently, Pf matches Req if the sum of set bit values of Req (we denote it by $|Req|$) equals to the sum of corresponding bit values in Pf. Due to this reason, we are only interested in the elements of a profile corresponding to the set bit positions of Req. As the first step of aggregation, we take out and sum up the encrypted elements of each profile in accordance with the set bit positions of Req. More formally,

$$A = \prod_{1 \leq i \leq p | Req_i = 1} EPf_i = Enc_{PK}(\sum_{1 \leq i \leq p | Req_i = 1} Pf_i * D + SS) \tag{3}$$

The second part of equality in Equation 3 holds due to utilization of an additively homomorphic encryption scheme. The *Server* performs this procedure for each profile of the group and obtains $A_1, ..., A_k$. Finally, the *Server* sums up A_j values and obtains

$$\begin{aligned} EAggGPf &= \prod_{j=1}^{k} A_j = Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + SS_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + \sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} SS_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + \sum_{1 \leq i \leq p | Req_i = 1} \underbrace{\sum_{j=1}^{k} SS_j}_{=0}) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i}) \end{aligned} \tag{4}$$

As it can be easily verified from Equation 4, $EAggGPf$ is the encryption of sum of bit values of profiles (Bloom filters) multiplied by their corresponding delimiters (D_j). *PSP* will employ delimiters to extract individual matching results. The aggregation procedure is summarized in Algorithm 2.

Algorithm 2 Aggregate($EPf_1, ..., EPf_k, Req$)

1: **for** $1 \leq j \leq k$ **do**
2: $\quad A = \prod_{1 \leq i \leq p | Req_i = 1} EPf_{j,i}$
3: **end for**
4: $EAggGPf = \prod_{j=1}^{k} A_j$
5: **return** $EAggGPf$

PSP: *PSP* decrypts the aggregated data (*PSP* possesses two secret keys SK_1 and SK_2. It may use one or both of the keys to obtain the plaintext data. Since *Server* is assumed to be honest but curious, the encryption under PK_1 is consistent with the encryption under PK_2). Then, *PSP* counts the number of profiles matched to the request by proceeding as follows. We denote the decryption of $EAggGPf$ by $AggGPf$, that is

$$AggGPf = \sum_{j=1}^{k} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{5}$$

Let us reformulate $AggGPf$ by extracting the first term of the outer summation as

$$AggGPf = D_k * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{k,i} + \sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{6}$$

Using Equation1, we know that

$$D_k > \sum_{j=1}^{k-1} D_j * p > \sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{7}$$

Therefore, if we divide $AggGPf$ by D_k we obtain the quotient and the remainder as indicated in Equation 8:

$$AggGPf = D_k * \underbrace{\sum_{1 \leq i \leq p | Req_i = 1} Pf_{k,i}}_{\text{Quotient}} + \underbrace{\sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i}}_{\text{Remainder}} \tag{8}$$

The Quotient is the summation of bit values of Pf_k in accordance with the set bit positions of Req. Thus, if the Quotient equals to $|Req|$, then Pf_k is a match. *PSP* continues the iteratively on the Remainder, using D_{k-1} for the next division. At any step that the number of target users exceeds the threshold, *PSP* sends *Yes* to the *Server* and stops. If it was not the case, *PSP* responds *No*. The process of matching is presented in Algorithm 3. Note that, the delimiters enabled us to extract the matching result of individual members from the aggregate, but *PSP* does not have any information regarding the individual profiles and usernames.

Algorithm 3 $Match(AggGPf, |Req|)$

1: $count = 0$
2: **for** $D_j \in DSet,\ j \in \{k, ..., 1\}$ **do**
3: $\quad$ **if** $\frac{AggGPf}{D_j} == |Req|$ **then**
4: $\quad\quad count = count + 1$
5: $\quad\quad$ **if** $count == Threshold$ **then return** *Yes*
6: $\quad\quad$ **end if**
7: $\quad$ **end if**
8: $\quad AggGPf = AggGPf \ \ mod \ \ D_j$
9: **end for**
10: **return** *No*

***Server* Show**: Based on the *PSP*'s response, *Server* either advertises the *Product* for all the members of the group, or skips that group. The total number of target groups is counted and stored for monetizing purposes.

For each advertisement request, the protocol above is repeated for each group that is not yet matched with the request. Since we prevent group compounding, matchings can be performed in *parallel*. This means, while the computational complexity scales, communication rounds do *not* need to increase with the number of yet unmatched groups.

V. Performance

A. Asymptotic Performance

Table Ia shows the computational overhead of each entity in PPAD based on the number of homomorphic operations. n corresponds to the total number of users in the OSN, and m

Overhead\Entity	User	Advertiser	*Server*	*PSP*
User registration	$O(p)$	-	-	-
Advertiser registration	-	O(1)	-	-
Advertisement	-	-	$O(k.\|Req\|)$	$O(1)$

(a) Running Time (per matching)

Overhead\Entity	User	Advertiser	*Server*	*PSP*
User registration	$O(p)$	-	$O(p)$	-
Advertiser registration	-	$O(p)$	$O(p)$	
Advertisement	-	-	$O(1)$	$O(1)$

(b) Communication Complexity (per matching)

TABLE I (a) Running time based on the homomorphic operations. (b) Communication complexities (number of message transmissions). k: number of users per group. p: size of Bloom filter. $|Req|$: number of set bits in each advertising request, which is $O(e.d)$ where e is the number of attributes in each request and d is the number of hash functions used in the Bloom filter construction.

corresponds to the total number of advertising requests.

Users The user carries $O(p)$ computational overhead, only once, to element-wise encrypt its Bloom filter under *PSP*'s public keys where p is Bloom filter's size.

Advertisers The advertiser does not perform any cryptographic operation.

Server The running time complexity of *Server* to aggregate users' profiles within their corresponding groups is $O(k.|Req|)$ where $|Req|$ is the number of set bits in the Bloom filter of the advertising request. *Server* carries this overhead per group and advertising request pair. In total, there are $\frac{n}{k}$ groups and m advertising requests, hence the total overhead of *Server* yields to $O(m.\frac{n}{k}.k.|Req|) = O(m.n.|Req|)$.

***PSP*:** For a single matching, *PSP* has the running time complexity of $O(1)$ (to decrypt the group aggregated profiles). *PSP* performs the matching procedure per group and advertising request pair, which in total leads to the complexity of $O(m.\frac{n}{k})$ for its lifetime.

Table Ib demonstrates the communication complexity of each entity during the execution of each protocol. Users and advertisers need to share their Bloom filters with *Server*. Thus, $O(p)$ message transmission is required. *Server* and *PSP* communicate $O(1)$ messages to check the matching between a single group and an advertising request. For n users ($\frac{n}{k}$ groups) the total communication overhead of *Server* and *PSP* is $O(\frac{n}{k})$.

B. Concrete Performance

The running times are computed by executing PPAD over 1000 randomly generated profiles of 400 attributes (based on our personal experience of Facebook advertising, 400 attributes is approximately the maximum number) under the group size of 5. The advertising request is presumed to have 30 attributes (for randomly generated profiles, almost no match is found for an advertisement with more than 30 attributes). The results are taken on an Intel i5 2.60 GHz CPU, using 2048 bit keys for Paillier encryption scheme. Under this configuration, *Server* matches a single advertising request to a single group in 50 ms whereas running time of *PSP* is 6 ms, which is almost an order of magnitude better than that of *Server*. Profile creation time is 750 ms (done once per user) and creating an advertising request takes 0.5 ms.

C. Advertisement Accuracy Metrics

In order to analyze the effect of different group sizes and threshold values on the advertising performance, we define two performance metrics, namely ***Target accuracy*** and ***Non-Target accuracy***.

Target accuracy indicates the fraction of target users who are served by the advertisement, as formulated in Equation 9

$$\textbf{\textit{Target accuracy}} = \frac{\text{Number of target users served by the advertisement}}{\text{Total number of target users}} \tag{9}$$

This metric is in compliance with the advertiser desire who wants to reach as many target users as possible. Due to the nature of group-based advertising, the *Target accuracy* is not always 100% since the target users in groups with fewer than threshold-many target users are not shown the advertisement.

Non-Target accuracy as shown in Equation 10 is the fraction of non-target users that are **not** served an (irrelevant) advertisement.

$$\textbf{\textit{Non-Target accuracy}} = \frac{\text{Number of non-target users not served by the advertisement}}{\text{Total number of non-target users}} \tag{10}$$

The higher value of this metric indicates that users are less likely to be shown irrelevant advertisement (hence more accurate is the advertising and less disturbing).

Note that the *Target accuracy* and *Non-Target accuracy* are meaningful only in the group-based advertising paradigm and not in personalized counterparts (where both measures are perfectly satisfied with the cost of privacy loss).

We additionally define the notion of **target coverage**, which is the fraction of target users, as follows:

$$\textbf{Target Coverage} = \frac{\text{Number of target users}}{\text{Total number of users}} \tag{11}$$

The coverage value depends on the attribute distribution in profiles as well as the content of the advertisement. In our experiments, we target various levels of coverage and analyze the effect of our system parameters.

D. Advertisement Accuracy Results

We explore the effect of group size and threshold value on the *Target accuracy* and *Non-Target accuracy*. The results are taken over 100,000 profiles with three different target coverage values (10%, 50% and 90%) as demonstrated in Figure 5. The results present that under a specific group size, increasing the threshold value improves the *Non-Target accuracy*. This behavior is expected since having a higher threshold guarantees that more target customers are in the target groups (compared to the lower thresholds). Hence in such settings, the higher percentage of target group members are real target customers i.e., the *Non-Target accuracy* is higher. On the contrary, the *Target accuracy* has the inverse relation with the threshold value. Indeed, higher threshold imposes more constraint on the group for being selected as a target. Consequently, the advertiser loses some of his target customers in the groups which do not have enough target users.

On the other hand, with a fixed threshold, as the group size increases, *Target accuracy* increases but *Non-Target accuracy*

decreases. This happens for all target coverages, since in a larger group with the same threshold, it is easier to find matching groups, but it also means that potentially more non-target users are shown an irrelevant advertisement.

By inspecting the behavior of *Target accuracy* and *Non-Target accuracy*, we find out that a perfect balance between these two metrics is met when the ratio of the threshold to the group size i.e., $\frac{Thr}{\text{Group Size}}$ is close to the target coverage. We refer to this threshold value as **"balanced threshold"**. For instance, under the target coverage 50% and group size 19, the balanced threshold is 10 with $\frac{10}{19} = 0.52 \approx 0.5$. At this balance threshold, *Target accuracy* and *Non-Target accuracy* are 58% and 60%, respectively. We refer to the accuracy achieved at the balance threshold by **balanced accuracy**. In Figure 5, the x coordinate of the point where two curves of the same color (i.e., same group size) collide indicates the balanced threshold and the accuracy at that point (y coordinate) is the balanced accuracy. After the balanced threshold, the *Target accuracy* drops while the *Non-Target accuracy* increases. The inverse occurs for values less than the balanced threshold.

The simulation results demonstrate that as the group size increases, the balanced accuracy degrades. For example, under the target coverage of 50%, the balanced accuracy of group size 7 (at balanced threshold 4) is 65% whereas in group size 19 (at balanced threshold 10) it drops to 58%. The correctness of this fact can be verified by coverage 10 and 90 as well. This implies that smaller group sizes are better for accuracy at their respective balanced thresholds.

In general, threshold being equal to group size k would mean that all users in a matched group have the same attributes in the advertisement in common. Similarly, threshold of 1 where the advertisement is not matched would reveal that no user in that group contains all the attributes in the advertisement. Such leakages are independent of the underlying methodology, and hence are not analyzed, but should be considered when selecting the parameters.

VI. Security

A. Security Definition

PPAD preserves user privacy if no adversary can link a successful group-matching result to a particular group member. In another word, the advertising result should not help an adversary to identify which user possesses (or does not possess) which attributes. The adversary controls *either Server* or *PSP* (since they are non-colluding), together with some users and advertisers. The adversary is challenged to break the user's privacy in a single group. This challenge is modeled as a game played between a challenger and the adversary A. Since the groups are independent of each other and the protocol is the same for every group, the failure of the adversary in this game implies that PPAD preserves privacy of all the users.

In this game, the adversary is allowed to control $k-t$ users where k is the group size and t is the number of honest users in that group. Adversary registers $k-t$ users of the group into the system and receives all of their secret information. Assume $UName_1$, ..., $UName_t$ are the usernames of the honest users. Adversary is asked to select t sets of attributes, $AttSet_1$,...,$AttSet_t$. The challenger randomly and privately assigns the attribute sets to the usernames and registers them into the OSN. Then, the adversary is allowed to register polynomially-many advertising requests and obtain the results of matching between the requests and the group. Finally, the adversary is challenged to guess which attribute set is assigned to which username. To win the game and break security, the adversary needs to perform noticeably better than the random guessing probability of $\frac{1}{t}$.

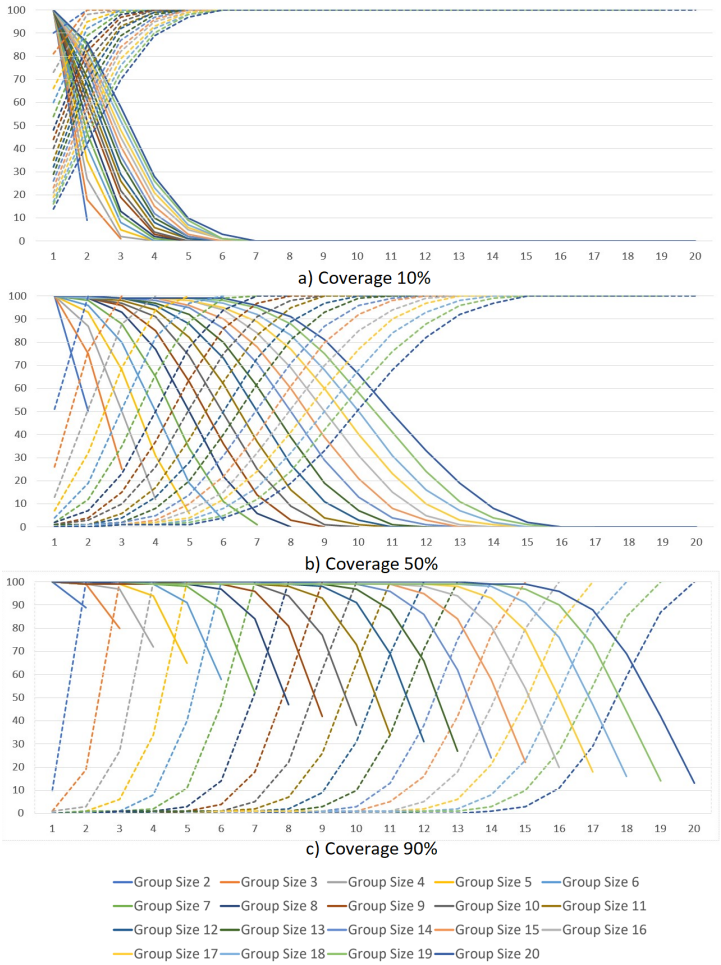


Fig. 5: *Target accuracy* and *Non-Target accuracy* vs threshold for group sizes 2-20. Dashed curves represent *Non-Target accuracy* and solid ones the *Target accuracy*. X axis: threshold. Y axis: accuracy

Observe that as t gets smaller, the adversary has more control over the group, and hence has more power. But, for $t = 1$, the adversary wins the game with the probability of 1; therefore $t = 2$ is the minimum feasible value.

$UPrivacy_A(\lambda)$: In this game, the challenger acts as the honest users and honest advertisers. One of the *Server* or *PSP* is run by the challenger while the other one is controlled by the adversary A. Adversary A is honest but curious, and may control polynomially-many advertisers and k-2 users per group. The game is played within one group.

1) A runs OSNInit with the challenger.
2) Query phase 1:

a) A runs $UReg$ protocol, acting as a user, with the challenger.
b) A specifies user's inputs $UName$,$AttSet$ and asks the challenger to run $UReg$ protocol over the given inputs. Challenger acts as user.

Part (a) allows the adversary register users fully under her control. Part (b) allows the adversary to register honest users whose usernames and profiles are known to the adversary.

3) Challenge phase:
   a) A sends two usernames i.e. $Uname_0$ and $UName_1$ and two different sets of attributes, $AttSet_0$ and $AttSet_1$. $UName_0$ and $UName_1$ are never registered in any query phase.
   b) Challenger picks a bit randomly, $b \leftarrow\{0,1\}$, and executes the $UReg$ protocol for ($Uname_0$, $AttSet_b$) and ($UName_1$, $AttSet_{\hat{b}}$) on behalf of users. $\hat{b}$ is the complement of b.
4) A repeats the query phase 1 until all the k users of the group are registered into the OSN.
5) Query phase 2:
   a) A creates and registers an advertising request by executing $AdReg$ protocol acting as the advertiser.
   b) A selects the inputs of the advertiser for $AdReg$ protocol and asks the challenger to execute $AdReg$ protocol as an advertiser.

   Similar to query phase 1, part (a) allows the adversary register advertisement requests fully under her control. Part (b) allows the adversary to register honest advertising requests of which are known to the adversary.
6) Query phase 3: A executes the Ad protocol with the challenger for an advertising request registered as RID.
7) A may adaptively repeat the query phase 2 and 3 polynomially many times.
8) A guesses a bit b'. If $b = b'$ the output of game is 1 (A wins), otherwise 0 (A loses).

Definition 6.1: An OSN with the ($OSNInit$, $UReg$, $AdReg$, Ad) protocols preserves the user's privacy, if for every probabilistic polynomial time (PPT) adversary A, there exists a negligible function $negl(\lambda)$, where λ is the security parameter, such that: $Pr[UPrivacy_A(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$

B. User Privacy Against Server

The security of our design against *Server* relies on the CPA-security of the underlying encryption scheme. We show that if a PPT adversary A can win the $UPrivacy$ game with non-negligible advantage, then we can construct a PPT adversary B who runs A as a subroutine and breaks the CPA-security of the encryption scheme. At a high level, since group matching does not reveal the identity of the targeted users, but only provides a *Yes/No* type answer, *Server* cannot map users' profiles and usernames using the result of advertising. Therefore, the information of *Server* is restricted to the encrypted data. Thus, the success of adversary A in $UPrivacy$ game implies that B can distinguish between the encrypted profiles of users. This means that encryption scheme is not CPA-secure which is a contradiction to the initial assumption. So, using a CPA-secure encryption scheme, our design is secure against *Server*. The formal proof is provided in the full version [1].

C. User Privacy Against PSP

PPAD provides information-theoretic privacy for users against *PSP*. *PSP* never receives the usernames during the execution of any protocol. This implies the inability of *PSP* to obtain a mapping between the data contents, i.e., attributes, and the identity of the data owners.

VII. Related Works

A. Secure Online Behavioral Advertising (SOBA)

In SOBA models, a broker is connected to a set of publishers who are web page owners. The broker creates a behavioral profile per user according to the user's visits on those pages. Broker monetizes by putting the advertiser's products on the publishers' web pages according to the users' behavioral profiles. We classify SOBA models as publisher-subscriber and push-based designs as shown in Table II.a (also considering PPAD applied to such a setting). In publisher-subscriber designs [34], [15], [13], users subscribe to the advertisers' products. In push-based designs [6], [2] a server receives both the users' profiles and advertising requests, and advertises each product for a set of target users. Some SOBA studies require users or advertisers to be online during the advertising procedure [34], [15], [6], while others allow them to remain offline [2]. Some studies [15], [6] enforce direct communication between users and advertisers. Outsourced profiling [6] does not consider user privacy. ObliviAd [2] relies on a trusted hardware (CPU) to protect user privacy.

B. Server Assisted Private Set Intersection (PSI)

In the PSI problem, two parties who have two different sets of elements execute a protocol to find the intersection of their sets. In the server assisted variant of PSI, a server helps the parties to find the intersection of the sets, improving efficiency. Table II.b summarizes the comparison between PPAD and papers of the server-assisted PSI concept. In the server-assisted PSI studies, the role of the server is to reduce the workload of parties by carrying the main portion of the computations. However, at least one party still needs to be involved per protocol execution as in [17], [29], [22] and the oblivious service provider method of [21]. Parties also need to have direct communication for sharing some secret information before the execution of the intersection (advertisement) protocol. The public output method of [21] and [36] support offline users and advertisers, but they fail to protect the privacy of users. In fact, their solution is vulnerable to the plaintext guess attack where the server guesses some elements and checks whether they belong to the user's and advertiser's sets or not.

C. Server Assisted Two-Party Computation (2PC)

In server assisted 2PC protocols, two parties, with the help of a third party, compute a function over their respective inputs while no party learns the other party's input. Server-assisted 2PC solutions either employ a server to guarantee the fairness of the protocol execution [14], [23], [25] or to ease the other parties' duties by delivering the main computation overhead to the server. However, users and advertisers are required to be online and provide some information per function evaluation

Type	Method	Offline User	Offline Advertiser	EDC	User Privacy	No IP Proxy	No Trusted-Hardware	Sec-Def
PS	Adnostic [34]	✗	✓	✓	✓	✓	✓	✗
	Targeted advertising [15]	✗	✗	✗	✓	✓	✓	✓
	Privad [13]	✓	✓	✓	✓	✗	✓	✗
PB	Outsourced profiling [6]	✗	✗	✗	✗	✓	✓	✗
	ObliviAd [2]	✓	✓	✓	✓	✓	✗	✓
	PPAD	✓	✓	✓	✓	✓	✓	✓

(a)

Method	Offline User	Offline Advertiser	EDC	PGA
Privacy aware Genome Mining [29], Scaling PSI to billion elements [17]	✗	✗	✗	✓
VDSI [36],Collision Resistant outsourcing PSI [21] public output	✓	✓	✓	✗
Collision Resistant outsourcing PSI [21] Oblivious Service Provider	✗	✗	✓	✓
Outsourced PSI using homomorphic encryption [22]	✓	✗	✓	✓
PPAD	✓	✓	✓	✓

(b)

TABLE II (a) SOBAs vs. PPAD. (b) Server Assisted PSIs vs. PPAD. PS: publisher-subscriber, PB: push-based. EDC: Elimination of direct communication between users and advertisers. Sec-Def: Existence of formal security definition and proof. PGA: Security against plaintext guess attack.

(advertisement in this case) [18], [20], [14], [26], [16], [10], [7]. [30] proposed a solution which mitigates the necessity of online users and advertisers by applying two servers, similar to our approach. But, the number of messages transferred between two servers depends on the function definition (number of multiplication operations), whereas PPAD supports constant communication compexity between two servers.

VIII. Conclusion and Future Work

In this paper, we proposed the first privacy preserving advertising system PPAD for secure OSNs with transparency and group advertising. PPAD protects users' privacy by employing an external non-colluding privacy service provider. We proposed a security definition and formally proved the security of our design under the honest-but-curious adversarial model where the adversary is additionally allowed to control some (fake) advertisers and users. As future work, our aim is to extend PPAD to be secure against fully malicious adversaries, and to efficiently support any Boolean function of the attributes in a single advertising request. We also plan to improve our solution to reduce the computational cost associated with profile updates. We believe PPAD constitutes an important first step regarding monetization for secure OSNs.

Acknowledgements

We acknowledge the support of the Turkish Academy of Sciences, Royal Society of UK Newton Advanced Fellowship NA140464, and EU COST Action IC1306.

References

[1] https://archive.org/details/staheri14_ku_PPAD.
[2] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *Security and Privacy (SP)*. IEEE, 2012.
[3] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM*, 2009.
[4] A. Barenghi, M. Beretta, A. Di Federico, and G. Pelosi. Snake: An end-to-end encrypted online social network. In *ICESS*. IEEE, 2014.
[5] A. Beimel. Secret-sharing schemes: a survey. Springer, 2011.
[6] D. Biswas, S. Haller, and F. Kerschbaum. Privacy-preserving outsourced profiling. In *CEC*. IEEE, 2010.
[7] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PET*, 2016.
[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. 1970.
[9] A. A. Bruen, M. A. Forcinito, A. G. Konheim, C. Cobb, A. Young, M. Yung, and D. Hook. Applied cryptography: protocols, algorithms, and source code in c. 1996.
[10] H. Carter, B. Mood, P. Traynor, and K. Butler. Outsourcing secure two-party computation as a black box. In *Cryptology and Network Security*. 2015.
[11] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of twitter. In *Security and Privacy (SP)*. IEEE, 2012.
[12] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with frientegrity: Privacy and integrity with an untrusted provider. In *USENIX*, 2012.
[13] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *NSDI*, 2011.
[14] A. Herzberg and H. Shulman. Oblivious and fair server-aided two-party computation. *Information Security Technical Report*, 2013.
[15] A. Juels. Targeted advertising... and privacy too. In *CT-RSA*. 2001.
[16] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011.
[17] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *FC*. 2014.
[18] S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *CCS*. ACM, 2012.
[19] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC press, 2014.
[20] F. Kerschbaum. Adapting privacy-preserving computation to the service provider model. In *CSE*. IEEE, 2009.
[21] F. Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *Applied Computing*. ACM, 2012.
[22] F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *CCS*. ACM, 2012.
[23] H. Kılınç and A. Küpçü. Efficiently making secure two-party computation fair. In *FC*, 2016.
[24] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *WOSN*. ACM, 2008.
[25] A. Küpçü and P. Mohassel. Fast optimistically fair cut-and-choose 2pc. In *FC*, 2016.
[26] P. Mohassel, O. Orobets, and B. Riva. Efficient server-aided 2pc for mobile phones. *PET*, 2015.
[27] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Security and Privacy*. IEEE, 2009.
[28] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.
[29] C. Patsakis, A. Zigomitros, and A. Solanas. Privacy-aware genome mining: Server-assisted protocols for private set intersection and pattern matching. In *CBMS*. IEEE, 2015.
[30] A. Peter, E. Tews, and S. Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE T-IFS*, 2013.
[31] J. Sun, X. Zhu, and Y. Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *INFOCOM*. IEEE, 2010.
[32] S. Taheri-Boshrooyeh, A. Küpçü, and Ö. Özkasap. Security and privacy of distributed online social networks. In *IEEE ICDCSW*, 2015.
[33] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: better privacy for social networks. In *CoNEXT*. ACM, 2009.
[34] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.
[35] T. Veugen, R. de Haan, R. Cramer, and F. Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE T-IFS*, 2015.
[36] Q. Zheng and S. Xu. Verifiable delegated set intersection operations on outsourced encrypted data. In *IC2E*. IEEE, 2015.

Is There a Case for Parallel Connections with Modern Web Protocols?

Jawad Manzoor[1], Ramin Sadre[1], Idilio Drago[2], and Llorenç Cerdà-Alabern[3]

[1]Université Catholique de Louvain, [2]Politecnico di Torino, [3]Universitat Politècnica de Catalunya

***Abstract*—Modern web protocols like HTTP/2 and QUIC aim to make the web faster by addressing well-known problems of HTTP/1.1 running on top of TCP. Both HTTP/2 and QUIC are specified to run on a single connection, in contrast to the usage of multiple TCP connections in HTTP/1.1. Reducing the number of open connections brings a positive impact on the network infrastructure, besides improving fairness among applications. However, the usage of a single connection may result in poor application performance in common adverse scenarios, such as under high packet losses. In this paper we first investigate these scenarios, confirming that the use of a single connection sometimes impairs application performance. We then propose a practical solution (here called H2-Parallel) that implements multiple TCP connection mechanism for HTTP/2 in Chromium browser. We compare H2-Parallel with HTTP/1.1 over TCP, QUIC over UDP, as well as HTTP/2 over Multipath TCP, which creates parallel connections at the transport layer opaque to the application layer. Experiments with popular live websites as well as controlled emulations show that H2-Parallel is simple and effective. By opening only two connections to load a page with H2-Parallel, the page load time can be reduced substantially in adverse network conditions.**

***Index Terms*—HTTP/2, QUIC, MPTCP, Performance, Measurements**

I. Introduction

The web has become an essential part of our daily lives. We see a continuous trend of migrating traditional applications to the cloud, e.g., Microsoft Office 365 and Google Apps. As a result, modern web content has become extremely complex. This complexity requires efficient web delivery protocols to maintain users' experience regardless of the technology they use to connect to the Internet and despite variations in the quality of users' Internet connectivity.

HTTP, which is the de facto standard protocol of the web was developed in early 1990s as a simple request/response protocol to deliver content over the Internet. Its first versions, HTTP/1.0 and HTTP/1.1, have inherent inefficiencies when dealing with modern web content. For example, they suffer from head-of-line (HOL) blocking, where responses must arrive sequentially, following the order of requests. As web pages were getting more and more complex over the years, these inefficiencies started to hurt Page Load Time (PLT).[1] Despite these limitations, HTTP/1.1 over TCP has maintained a dominant position for around 20 years due to well-known challenges in replacing popular Internet protocols.

[1]Page Load Time is the time from when a user fires a web page request (e.g., by clicking on a link) until the page is fully loaded by the browser.

Browser vendors have reacted to HTTP/1.1 inefficiencies throughout the years by deploying ad-hoc optimizations to speed up PLT. One such optimization is the opening of several persistent TCP connections towards each web server when retrieving pages. Browsers can issue requests in parallel in the multiple connections, reducing the effect of HOL blocking. As a side effect, they compete for resources with other applications in the network more aggressively. For example, while the transfer rate of a single TCP connection is limited by the small congestion window (cwnd) during TCP slow start, multiple connections sum up their cwnd, resulting in faster startup rates. The fierce competition among browser manufacturers has pushed browsers to open a large number of parallel connections in an attempt to speed up page rendering [1].

Only recently, with Google's development of SPDY and QUIC, new protocols to replace HTTP/1.1 have gained momentum. The successful deployment of SPDY over TLS has opened the way for the HTTP evolution, triggering the standardization of HTTP/2 [2]. HTTP/2 borrows many of SPDY's principles and solves several shortcomings of HTTP/1.1. In particular, HTTP/2 *multiplexes* requests in a single TCP connection, eliminating the HOL blocking bottlenecks. This feature has prompted the IETF to recommend clients to open a single TCP connection per host-port pair for HTTP/2 transactions [2].

QUIC (Quick UDP Internet Connections) is another promising protocol developed by Google that provides multiplexing, congestion control and security functionality similar to HTTP/2, TCP and TLS, respectively, on top of UDP. It implements several optimizations including 0-RTT connection establishment, where clients can start repeated sessions with a known server without a three-way handshake, improved congestion control and better RTT estimation and loss recovery mechanism than TCP [3].

Recent studies have tracked the adoption of HTTP/2 and QUIC, showing not only a manifold increase in their usage, but also real performance gains [4], [5], [6]. However, both protocols use a single connection by design, which may result in poor application performance under adverse network conditions, in particular if different protocols compete for resources. For example, HTTP/2 is known to be particularly vulnerable in WiFi networks with high random packet losses. Equally, whereas QUIC uses a different congestion control strategy that reduces the effects of random packet losses, the implications

ISBN 978-3-903176-08-9

of QUIC's use of a single connection – e.g., during congestion in load-balanced links – are not fully understood yet.

In this paper we investigate the performance of browsing using modern web protocols in some adverse network scenarios. We use both active measurements with live websites and emulations in a testbed. We first confirm that HTTP/2 (and to a lesser extent QUIC) suffers more than HTTP/1.1 with multiple TCP connections in the tested scenarios. The use of a single connection partly explains the results. We then test whether adopting multiple TCP connections with HTTP/2 helps in mitigating the problems. We call this practical solution H2-Parallel, and implement it by modifying the source code of the Chromium browser. We compare H2-Parallel with HTTP/2 using a single TCP connection, HTTP/1.1 (both cleartext and encrypted) using multiple TCP connections, QUIC over a single UDP connection, as well as HTTP/2 over Multipath TCP (hereafter called H2-MP). The latter creates parallel connections at the transport layer opaque to the application layer. Note that HTTP/2 and QUIC always employ encryption by default.

Our experiments with popular live websites as well as controlled emulations show that H2-Parallel has some interesting advantages. It reduces PLT when compared to HTTP/2 over a single connection. In a scenario with around 2% of packet loss, H2-Parallel with only two parallel connections reduces the average PLT of HTTP/2 by 55%, and practically makes the performance of HTTP/2 similar to what is obtained by HTTP/1.1 with several parallel connections. Although QUIC is not affected by packet losses as severely as HTTP/2 over TCP thanks to its new congestion control strategy, we show that QUIC can also benefit from the use of parallel connections in some scenarios. Finally, H2-Parallel and H2-MP present similar performance with different practical trade-offs.

We make the following contributions:

- We identify scenarios that challenge HTTP/2 and QUIC performance, namely (i) packet losses in wireless networks and (ii) congestion in ISP networks with load balancers, a scenario not addressed in prior work yet;
- We implement *H2-Parallel*, a Chromium-based user agent that fans out HTTP/2 requests to a destination over multiple TCP connections;
- We compare *H2-Parallel* against the major web protocols. Our results differ from previous works by (i) including all relevant web protocols, instead of only a subset of them; (ii) considering real websites and browsers instead of simplistic downloads or TCP transfers, thus giving a view on how users perceive performance while browsing; (iii) testing the latest protocol versions (e.g., QUIC 39).
- We evaluate whether the protocols are fair to one another when competing for bandwidth and find that *H2-Parallel* and QUIC behave similarly for long transfers.

To simplify the terminology, in the remainder of this paper, we will refer to the non-encrypted (i.e., cleartext) version of HTTP/1.x as **H1C**, to HTTP/1.x over TLS as **H1**, and to HTTP/2 over TLS as **H2**.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III we discuss scenarios in which H2 and/or QUIC may exhibit poor performance. We present our measurement methodology in Section IV, and discuss results in Section V. Section VI concludes the paper.

II. Related work

Zimmermann et al. [5] investigate H2 adoption. They show that around 12.5% of Alexa top-million domains provide full H2 support, with a 66% increase in H2 enabled domains between Sep 2016 and Jan 2017. They also study H2 performance, but in contrast to our work, they focus on the impact of H2 server push functionality, showing that some websites profit from the feature to speed up PLT.

Other works characterize the performance of websites according to the used HTTP versions. Zarifis et al. [7] explore the PLT differences between H1 and H2 using data collected from real users of the Akamai CDN. They find that in around 60% of the time H2 has lower PLT than H1. Varvello et al. [4] build a measurement platform to actively monitor H2 adoption by probing Alexa top-million websites. They show that around 80% of the websites adopting H2 improve PLT.

Saxcé et al. [8] compare the performance of H1 and H2. They clone the Alexa top-20 websites and find that, apart from network conditions, PLT depends on the website structure and content. Erman et al. [9] focus on mobile browsing. They measure PLT for the top-20 Alexa websites using SPDY and H1 proxies in a 3G network and find that SPDY performs poorly due to the large number of retransmissions and TCP backoff. Elkhatib et al. [10] reach similar conclusions by comparing the performance of SPDY with H1 in simulated networks. All these works however do not explore possible solutions for the scenarios where H2 performance degrades.

Wang et al. [11] perform experiments with synthetic pages and cloned pages (Alexa top-200). They propose a solution for SPDY inefficiencies by tuning TCP: increasing initial window, increasing receive window and reducing backoff rate in case of packet loss. This solution is not practical since making changes to TCP is hard and can take years to be widely deployed.

Recently, there has been a lot of interest in QUIC. Carlucci et al. [12] investigate QUIC (v. 21) on emulated network environments using synthetic pages. They report that QUIC performs worse than H1, but better than SPDY, with large web pages and 2% random packet loss rate. Without packet loss, QUIC performs better than H1 and SPDY for small and medium web pages, but worse for large pages due to the usage of only six parallel streams.

Megyesi et al. [13] test QUIC (v. 20) while emulating different values for bandwidth, delay and packet loss. They host four synthetic pages having different size and number of images. They show that, with packet loss, SPDY performs the worst, followed by QUIC and H1. In case of high bandwidth and large page size, QUIC's PLT is three times larger than H1 and SPDY. Kakhki et al. [14] show the root-cause for the problem, which prevented slow start threshold update (fixed in newer QUIC versions). The authors show that QUIC (v. 34)

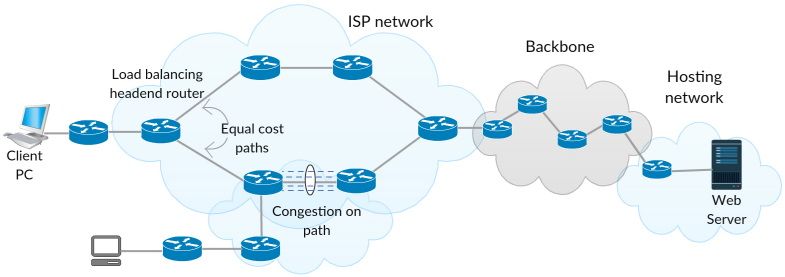


Fig. 1: Congestion on one of the load-balanced paths in ISP network

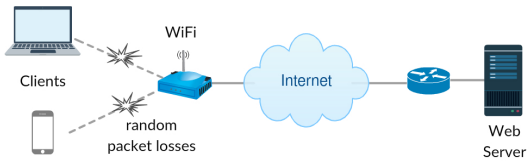


Fig. 2: Random packet losses in WiFi network

always outperforms H2, except with a very large number of small objects. QUIC (v. 34) is however unfair with other protocols, taking more than 50% of the bottleneck bandwidth when competing with 2 or even 4 TCP connections, a result that we will revisit with the newest version of QUIC.

In our prior work [15] we showed that some browsers arbitrarily create up to 6 connections towards a destination when using H2, but this was due to an issue only recently discovered by the developers [16] (fixed in Chromium v. 61), and not a performance improvement feature. In this paper we implement such a solution for H2 (i.e., *H2-Parallel*) and show its effectiveness.

Other studies have investigated the use of newer transport protocols such as MPTCP to alleviate the performance degradation of H2 and SPDY in presence of packet losses. Han et al. [17] provide the first measurement study of mobile web performance over MPTCP using SPDY and H1. They download 25 websites from Alexa top-100 list and measure PLT by combining LTE and WiFi with MPTCP. They show that MPTCP helps in mitigating performance penalties of SPDY under packet loss. In [18] the same authors provide a cost-benefit analysis of MPTCP in terms of improved user experience and energy consumption on mobile devices. The assumption in these studies is that clients are multi-homed – e.g., WiFi and LTE can be used simultaneously. Our work has a different scope: we check whether MPTCP has an impact on H2 performance even on single-homed devices, e.g., laptops or PCs with WiFi only. We will show that MPTCP can create multiple TCP connections and help in reducing the impact of packet losses and congestion in the network.

III. Considered scenarios

We consider real-world scenarios where H2 and QUIC may perform poorly. These scenarios are described in the following and investigated in the next sections.

A. Random packet losses in WiFi networks

Random packet losses in WiFi networks are common under real-world conditions due to various factors like noise and the distance from clients to the access point. The impact of packet loss on the performance of H1, SPDY and H2 has been thoroughly studied in prior work [10], [11], [8], [17]. There is a consensus that H2 performs poorly compared to H1 when there is high packet loss rate. We investigate the extent to which other protocols suffer from similar problems, and propose a practical solution applicable to regular H2 over TCP.

B. Load balancers and congestion in ISP networks

A large percentage of the Internet traffic between source-destination pairs traverses multiple paths due to deployment of load-balancing routers [19], [20] in ISP networks. These routers split packets across multiple paths using techniques like Equal-Cost Multipath Routing (ECMP). Since traditional Internet measurement tools like traceroute may fail to identify these paths, alternative tools such as Paris traceroute have been developed to quantify multipath routing in the Internet. Augustin et al. [21] performed a large scale measurement using over 68 thousand destinations and showed that around 70% of paths between source and destination networks traverse a load balancer.

Three different load-balancing schemes exist: packet-based, destination-based and flow-based. Packet-based algorithms distribute all incoming packets evenly on all network paths, e.g., in a round robin fashion. Since different paths can have different delay, this approach may result in massive packet reordering and hence out-of-order delivery of the packets [22]. Therefore it is rarely used in practice. The destination-based scheme routes all traffic destined to the same host over the same path. This scheme can lead to uneven traffic distribution. The flow-based scheme is more popular. It defines a flow using different fields in the packet header, such as source and destination IP addresses, port numbers and protocol. All packets belonging to the flow according to the chosen definition are sent over the same path.

However, optimal load balancing is hard. Figure 1 shows a load-balancing headend router with two equal-cost paths in an ISP network where one of the paths is shared by traffic from other nodes. Despite evenly distributing traffic using ECMP on the headend router, the shared link may become overloaded and cause congestion. A number of large scale measurements [23], [24], [25], [26] show that congestion predominantly occurs in ISP networks and the described scenario happens quite often. In such a scenario H2 and QUIC may perform poorly as the browser will open a single connection, and that connection may be routed over the congested path. Hence they cannot exploit the underlying path diversity of

the network. H1 on the other hand establishes multiple connections which may be distributed across the available paths by the load-balancing routers. Therefore, the performance is not significantly affected while downloading pages with H1 in such a scenario. This is an important scenario, yet it has not been investigated in prior studies. To quantify the extent of severity of this scenario by performing experiments in the Internet is an interesting topic for future work, but is out of scope for this paper.

C. Fairness among competing connections

In the recent years the traffic share of H2 and QUIC has rapidly increased and today, connections belonging to H1, H2 and QUIC co-exist in web traffic. These connections essentially compete for the bottleneck bandwidth. Maintaining fairness is very important for the network as unfairly taking bandwidth share from other protocols may lead to substantial performance degradation for some applications. While it is clear that H1 is unfair to H2 because of its aggressive use of connections, it is more interesting to see how QUIC competes with H1 and H2. In a recent study [14] on QUIC (v. 34) it was shown that QUIC is unfair to 2 and even 4 competing TCP connections. Since QUIC is evolving rapidly with major changes and improvements in each new version, we want to observe whether this behavior has changed in the most recent version (v. 39). Moreover, we want to verify how QUIC compares to our H2-Parallel implementation.

IV. Methodology

We now explain the design of H2-Parallel and H2-MP, our testbed, and measurement of PLT and cwnd of TCP and QUIC.

A. H2-Parallel

H2-Parallel is our Chromium-based user agent that fans out H2 requests over parallel TCP connections to mitigate the negative impact of a single connection on H2 PLT. Our objective is to verify that allowing the user agent to open parallel TCP connections for H2, similar to what most user agents do for H1, would improve the PLT.

Chromium browser maintains a single H2 session per domain, in accordance with the H2 specification. H2 sessions are tracked by a key consisting of the destination host–port pair. Each H2 session goes on a separate socket. In order to allow *two* TCP connections per domain, we have modified Chromium such that it stores two keys for each host-port pair. When issuing a new request, the state of the H2 session is controlled. First we check if we already have two keys for destination host-port pair of the current request. If not, we create a session with the new key and initialize the TCP connection. For each subsequent request, we call a function that returns one of the two available connections and use it for the request.

To keep the modifications as straightforward as possible, we have implemented a basic scheduler that assigns requests in a round robin fashion to one of the two connections. Note that requests assigned to the same connection are still multiplexed by H2. For the server, the two connections look like two regular H2 sessions from the same source IP. Despite this approach being simple, it distributes the requests fairly equally over the two connections.

TABLE I: Statistics of cloned web pages. The columns *HTML*, *CSS*, etc. show the number of objects of that respective type.

Website	HTML	CSS	JS	Image	Other	Total	Size (kB)
Baidu	1	1	1	6	1	10	50
Google	2	1	3	5	1	12	56
Live	2	2	2	2	0	8	262
Twitter	6	1	4	2	3	16	421
Wikipedia	1	1	2	20	1	25	441
Reddit	4	2	5	26	2	39	470
Yahoo	16	13	5	48	4	86	839
VK	4	1	14	3	1	23	920
Taobao	2	2	7	38	4	53	1 320
Instagram	3	1	7	25	1	37	1 409
QQ	15	6	19	115	6	161	1 728
Sohu	13	11	33	167	4	228	2 056
YouTube	8	3	5	113	20	149	2 911
Facebook	1	1	8	123	1	134	3 560
Amazon	5	2	14	41	2	64	3 723

TABLE II: Statistics of the pages in live websites

Website	Objects	Size (kB)	Domains	Connections		
				H2	H2-Parallel	H1
Google	17	286	1	1	2	4
Bing	32	421	1	1	2	2
Wikipedia	36	882	2	2	4	4
Mozilla	37	931	2	2	4	8
Poloniex	19	1 028	2	2	4	7
Paypal	64	1 415	2	2	4	12
Instagram	35	1 785	3	3	6	14
Blogger	61	2 061	2	2	4	11
Twitter	18	2 429	2	2	4	5
Facebook	86	4 266	2	2	4	12

B. H2-MP

H2-MP uses MPTCP to create parallel subflows to the servers to load a web page. We use H2-MP to compare the performance of creating parallel connections at transport layer versus application layer(as implemented by H2-Parallel). MPTCP is an enhancement of TCP that allows bandwidth aggregation and improved reliability by utilizing multiple paths simultaneously. It provides the same socket interface as TCP and spreads the data across several subflows without requiring applications or upper-layer protocols to be aware of the multiple paths. An MPTCP connection is initiated with the usual TCP 3-way handshake over one path. The handshake however includes a `MP_CAPABLE` message in the options field of the `SYN`, `SYN/ACK` and `ACK` packets. Further (sub-)flows can be added to the MPTCP session by sending `MP_JOIN` in the option field of additional 3-way handshakes regardless of the path used to open the flow.

We use stable release v0.91 of MPTCP and use the *ndiff-ports* path manager with the number of subflows set to 2, which creates two subflows between the same pair of IP-addresses by modifying the source port. We set the *default* scheduler which starts by sending data on the subflow with the lowest RTT. When its cwnd is full, it sends data on the subflow with the next lowest RTT. This is the recommended scheduler as it is known to provide the best performance. Hence two TCP connections are established between client and server without any modification of the browsing applications and having a complete view of the state of the connections at the transport layer. However, it requires MPTCP-compatible network stacks in the client and in the server.

C. Mininet testbed setup

We use Mininet [27] version 2.3 to emulate the three scenarios described in Section III. Mininet emulates a large network comprising multiple hosts, links and switches running real kernel and application code. We run Ubuntu Linux kernel 4.1.38 with the stable release v0.91 of MPTCP on the client and server and use Cubic congestion controller on both sides. We use Chromium browser version 60 on the client which supports H1, H2 and QUIC(v. 39). The server node hosts H2O web server[2] which provides an open-source implementation of H2, and quic-go web server which is an implementation of the QUIC protocol in Go[3]. We use Linux's Traffic Control *(tc)* and Network Emulation tools to configure network path characteristics such as bandwidth, delay and packet loss.

1) Emulating ECMP and congestion: For the scenario of ECMP with congestion, we emulate a typical home network shown in Figure 1 where a client's home router is connected to the headend router of an ISP with a 10 Mbps link [28]. The link from the ISP to the web server is configured with 1 Gbps bandwidth capacity. The headend router at the ISP performs load balancing using two paths. We emulate the congested bottleneck link in the lower path by generating traffic on it with 90% of its bandwidth capacity using iPerf with 8 connections.

We use flow-based routing in the load-balancing router for the reasons explained in Section III-B. Flow-based ECMP routing is not available in the latest MPTCP-capable Linux kernel that we use in our experiments. Therefore, we build a custom Linux kernel and implement a flow-based routing algorithm where the next hop is selected by hashing the flow 5-tuple, i.e., source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol type (PT) of a connection. Our hash function H is defined as

$$H = SA \oplus DA \oplus SP \oplus DP \oplus PT$$

where $\oplus$ is the bitwise XOR function. We calculate H mod 2 to select either the first path or the second path.

This design avoids any informed decision at the router on how the multiple flows of a single H1, H2-Parallel or MPTCP session are routed through the paths. For instance, the MPTCP or H2-Parallel flows may all take the congested or the non-congested paths during emulations. Obviously, each protocol will react to path choices differently. For instance, MPTCP is able to detect congestion and move traffic to non-congested paths, if at least one subflow is routed to the non-congested path. H2-Parallel instead will blindly schedule requests on the multiple connections.

2) Emulating random losses in WiFi: We emulate the network shown in Figure 2. The WiFi link has 7 Mbps bandwidth and 50 ms delay representing realistic network conditions based on large-scale measurement study [29] and also used in prior studies [17], [18]. We perform tests without and with packet loss in the WiFi link. For the latter, we inject random packet losses using *netem*. We use 2% packet loss rate as suggested in prior work [11], [12], [13]. We also perform experiments using other loss rates but the results are not shown due to space limitation.

D. Measuring page load time

We have selected 15 websites from Alexa's top 100 list and downloaded their landing pages or other publicly available pages onto the H2O server. The selected websites are a mix of social networking, online shopping, news and search. The main characteristics of the cloned pages are summarized in Table I. Chromium browser is used on the client to load the pages from the server. We configure dnsmasq [4] on the client to ensure that all hostnames resolve to the IP address of the server and do not leave the testbed. We have also selected 10 popular H2 enabled websites for *live* experiments listed in Table II. The key requirement for this selection is that *all* of the content must be delivered by the server using H2.

To automate the page loading we create a script that uses *Chrome-HAR-capturer*[5] to connect to the browser via its remote debugging API and load each page multiple times with cold cache. When the experiment ends, an HTTP Archive (HAR) file is created, containing detailed performance data. Our script parses the HAR file, extracts PLT for each of each run and calculates the arithmetic mean of all runs. We load the web pages using H1C, H1, H2, H2-Parallel, H2-MP and QUIC.

E. Measuring cwnd changes

We monitor the changes in the cwnd size for TCP using the *tcpprobe*[6] module. In case of H1 we calculate the sum of the cwnd sizes of all individual connections. In case of QUIC we instrument the source code of quic-go web server to collect logs that allow tracking of the cwnd size on each ACK.

V. Measurement Results

A. Impact of packet loss on single connection

We start our experiments by determining the impact of packet losses on the performance of various protocols by monitoring changes in cwnd size while loading a web page.

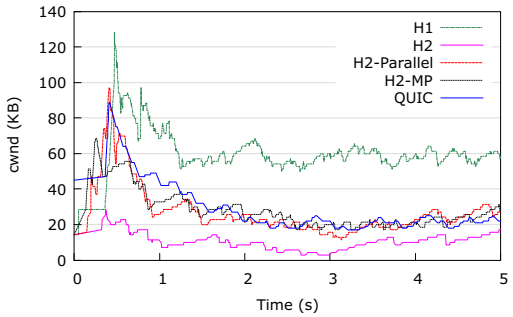
[2]https://h2o.examp1e.net/

[3]https://github.com/lucas-clemente/quic-go

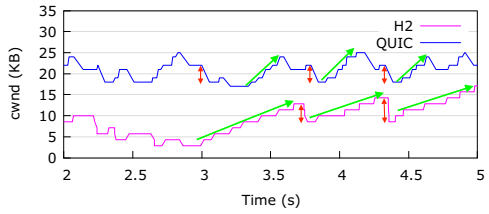
[4]http://www.thekelleys.org.uk/dnsmasq/doc.html

[5]https://github.com/cyrus-and/chrome-har-capturer

[6]https://wiki.linuxfoundation.org/networking/tcpprobe



(a) Comparison of cwnd of all protocols



(b) 3-second zoom comparing cwnd of H2 and QUIC

Fig. 3: Timeline showing the impact of packet losses on congestion window size during a page load

We perform an experiment where we host a static web page comprising several JPG images on our web server and load it on the client using Chromium browser via H1, H2, H2-Parallel, H2-MP and QUIC. We configure the bottleneck link with 7 Mbps bandwidth, 50ms RTT, 45 kB buffer size and inject 2% random packet losses into the network path using *tc* and *netem*. We log the changes in cwnd sizes.

Figure 3a shows a 5-second zoom of the cwnd size comparison of the protocols. We can see that H1 has a much higher cumulative cwnd size than others. This is because browsers usually maintain up to 6 parallel connections to each server for H1 transfers and only some connections may be affected by random losses at a time. So the sum of cwnd size of all individual connections remains high. On the other hand, since browsers establish only one connection to the server when using H2, the same connection keeps experiencing the losses resulting in continuous reduction of cwnd. This limits the cwnd to a very small size which in turn results in very low throughput and long page load time. We can also see that QUIC has almost twice the size of cwnd as compared to H2 although both use a single connection and face the same rate of packet losses. There are several reasons for this behavior. First, note that the initial window size in QUIC is around 45 kB (32 segments) while for H2 (and others based on TCP) the size is around 15 kB (10 segments). Second, QUIC has an advantage over H2 because it uses its congestion controller to emulate the behavior of two TCP connections over UDP (in QUIC version 39). In other words, in the event of packet loss the cwnd is reduced at half the rate of H2, depicted by red double-headed arrows in Figure 3b. Finally, QUIC recovers more quickly from packet losses than H2, which can be observed by a steep upslope as shown by green arrows in Figure 3b.

In case congestion in network with load-balancers, the packet losses are dynamic, however, the results that we observe are almost the same and are not shown here. In such a scenario, the single connection of H2 and that of QUIC suffers losses when it is on the congested path, while H1, H2-Parallel and H2-MP are able to use the non-congested path simultaneously for part of their traffic.

B. Impact of packet loss in WiFi networks

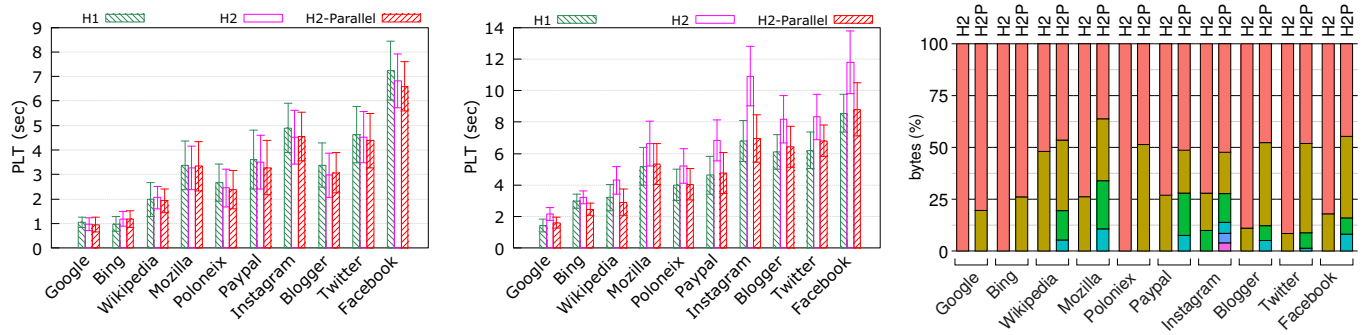
1) Packet loss in live websites: When visiting a live website, several aspects influence the PLT perceived by the user, e.g., delays of DNS queries or of server-side operations to prepare the content. Furthermore, a live website might consist of objects coming from different domains (e.g., due to domain sharding) that are not delivered from the same host.

We study the performance of H1, H2 and H2-Parallel with live websites. We do not perform experiments with MPTCP since none of the top websites support it at the server side. We also skip QUIC as it is only available for Google services and we cannot compare it with other websites. The only parameter that we will vary in our experiments with live websites is the random packet loss rate. To this end, we have selected 10 H2 enabled websites. The page characteristics are shown in Table II, where we give the number of objects per tested page, the total size in kBytes and the number of domains delivering the objects. The latter determines the number of TCP connections opened by the browser (also shown in the table). For each domain Chromium opens one TCP connection with H2, two with our H2-Parallel implementation, and up to six connections with H1. Note that the pages loaded from live websites are not exactly identical to those we cloned onto our testbed.

The RTT to the web servers is in the 15–165 ms range. We load each page 15 times with an empty cache. Figure 4 shows the average PLT (and its standard deviation) when using H1, H2 and H2-Parallel without packet loss and with 2% packet loss.

Focusing on Figure 4a, we notice how the performance of H2 is mostly better than H1. Whereas differences are not extremely large, these results are significant if we consider the number of TCP connections opened by the browser for each protocol (see Table II). Our implementation of H2-Parallel achieves similar performance as H2 when network conditions are good, although a small overhead for opening and managing the extra TCP connections are visible in some cases.

Obviously, the PLT increases significantly when packet loss is introduced – see Figure 4b (note the different scale of the y-axes). However, the increase of PLT with H1 is less pronounced than with H2, thanks to the use of multiple TCP connections by the former. H2 suffers severely under the packet loss. We notice how the PLT for Facebook reaches almost 12 s on average for H2, whereas it is around 8.5 s for H1 with 2% packet loss. The figure also shows that H2-Parallel achieves similar performance to H1 thanks to its second TCP connection. We are able to achieve 53% reduction in PLT on average for all websites by using H2-Parallel.

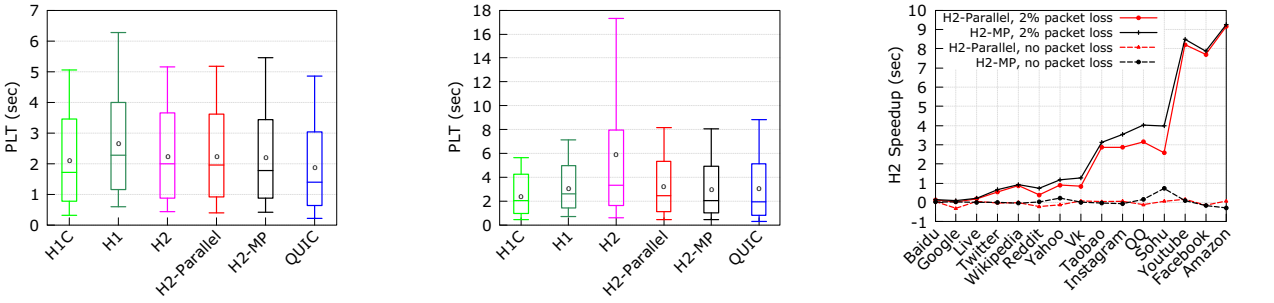


(a) Average PLT and std dev without packet loss

(b) Average PLT and std dev with 2% packet loss

(c) Percentage of data transferred on each connection with H2 and H2-Parallel (H2P)

Fig. 4: Performance comparison of live websites with and without packet loss

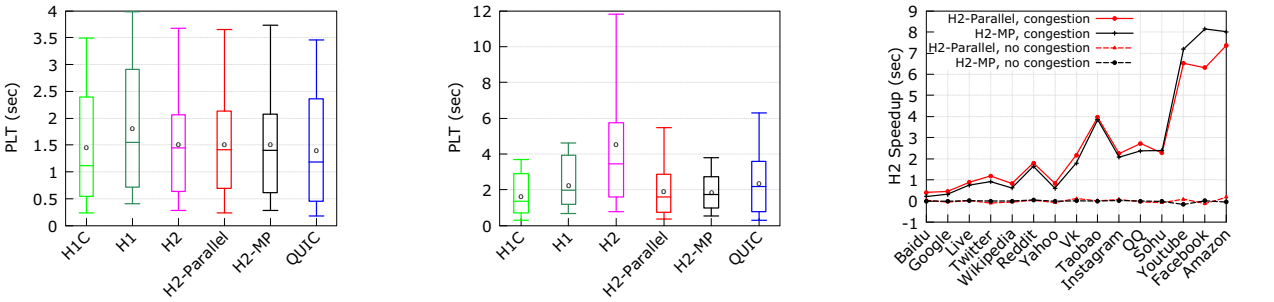


(a) WiFi with 0% packet loss rate

(b) WiFi with 2% packet loss rate

(c) H2 speedup against a single connection

Fig. 5: Emulation of WiFi network. Note differences in y-axes.



(a) No congestion on any path

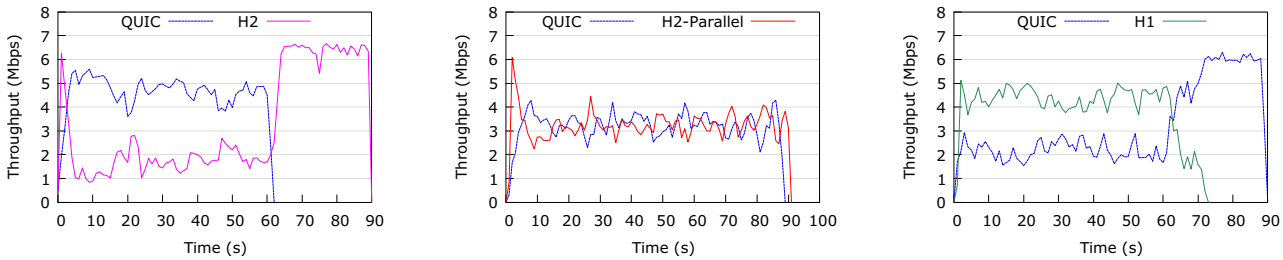
(b) Congestion on one path

(c) H2 speedup against a single connection

Fig. 6: Emulation of ECMP routing in the network. Note differences in y-axes.

Figure 4c shows the number of connections established with the server to load a website (depicted with different colors) and the percentage of bytes transferred with each connection (depicted by the size of the bar of each color) using H2 and H2-Parallel. We can see that with H2, while in some cases multiple connections are established due to multiple domains on the server side, most of the data (83% on average for all tested websites) is still transferred using only one connection. In case of H2-Parallel, a single connection carries 52% of traffic on average, thus distributing the load more evenly across multiple connections and reducing the probability of a single connection experiencing packet losses repeatedly.

2) Packet losses in emulated environment: We perform emulations in a controlled mininet environment for reproducibility of results. In this experiment we measure the effect of random packet losses on the performance of the different web protocols using cloned web pages. Since the web server is under our control we can test QUIC and MPTCP and compare them with other protocols for the same pages, which was not possible with live websites. We load each page 30 times with empty cache using automated scripts. Note that the performance of H1 and H2 in scenarios with packet losses in WiFi networks has been studied in [10], [8], [9], [11], [17]. We confirm results from the previous works and evaluate to what extent H2-Parallel and H2-MP improve performance.



(a) QUIC competing with H2 (b) QUIC competing with H2-Parallel (c) QUIC competing with H1

Fig. 7: Comparison of fairness of bandwidth share among competing connections of different protocols

In the following, box-whisker plots show the arithmetic mean (small circle), the median (middle horizontal line), the first and third quartiles (upper and lower box edges) and the minimum and maximum (whiskers) of the PLT over all 15 web pages. In our experiments, minimum and maximum will typically be several seconds apart. This is due to the large difference in characteristics of the selected web pages, i.e., small web pages like Google have very small PLT (represented by the lower end of the whisker) while large web pages like Facebook or Amazon have very large PLT (represented by the upper end of the whisker).

Figures 5a and 5b show plots of the PLT measured over all web pages without and with 2% packet loss, respectively. As expected, H2 performs better than H1 when there is no packet loss, and H2-Parallel and H2-MP do not show significant differences under good network condition. However, *with* packet loss, H2's PLT increases greatly while the other protocols see only a moderate increase by around 20%. Again, H2 is affected the most due to the use of a single TCP connection by the browser. Another disadvantage using H2 is that when there is a packet loss event, all streams get stalled until packet recovery due to the in-order delivery guarantee of TCP. Using QUIC, only the stream related to that packet gets blocked while others keep functioning normally. QUIC also maintains larger cwnd size as compared to H2 as shown in Figure 3a. Due to these reasons its performance is not affected as severely as H2.

Both H2-Parallel and H2-MP are able to reduce the performance penalty of packet losses and achieve a performance similar to H1 by increasing the cumulative cwnd size.

Figure 5c shows the average speedup (in seconds) that H2-MP and H2-Parallel achieve relative to regular H2 for each tested website (sorted by their size, with the smallest on the left). It can be seen that the speedup relative to regular H2 with packet loss is particularly pronounced for large web pages.

C. ECMP and network congestion in emulated environment

We now emulate the ECMP scenario with Mininet using H1C, H1, H2, H2-Parallel, H2-MP and QUIC. For each considered protocol, the client loads each web page 30 times with an empty cache. Remember that we do *not* actively control how the multiple flows are load-balanced in the available paths to emulate realistic scenarios. That is, in some experiment rounds, the multiple MPTCP or H2-Parallel flows may both take the congested or the non-congested path by chance. We can see in Figure 6a that, without congestion, H2 performs slightly better than H1 thanks to its various optimizations and new features. Not a surprise, H1C is faster than H1 because of the TLS overhead in the latter. Using two connections (H2-Parallel and H2-MP) in good network conditions brings no noticeable advantage while QUIC performs slightly better than others in the mean and median case.

However, the situation changes drastically in the presence of congestion. H2's PLT shoots up, with some pages taking as much as 12 s on average and up to 20 s in the worst case (not shown) to be fully loaded. H1C, H1, H2-Parallel and H2-MP are only slightly affected thanks to the parallel connections, which may be routed in the two available paths. In fact H2-MP performs the best as it can route the traffic away from the congested path on the fly and move it to the good path, which is not possible with any other protocol. QUIC is not affected as severely as H2 because its congestion controller reduces the cwnd size less aggressively when dealing with packet losses due to congestion. However, it still performs worse than H2-Parallel and H2-MP because in many cases it cannot take advantage of the non-congested path due to the use of a single connection. QUIC is 34% slower than H2-Parallel and H2-MP on average for medium and large websites but the situation is different in case of small websites. QUIC loads small websites quite fast due to 0-RTT connection establishment, and even with a single connection it performs slightly better than H2-Parallel and H2-MP. In fact for small websites creating parallel connections doesn't provide much benefit because most of the data is already transferred on the first connection before the second connection gets its turn.

Finally, in Figure 6c we can see that there is no speedup using H2-Parallel and moderate speedup using MPTCP when both network paths are congestion-free. When congestion is created on one path, both H2-Parallel and MPTCP achieve impressive speedups particularly for large pages.

Among the tested protocols, QUIC looks the most promising. Although it does suffer from performance degradation in the above scenario, we believe that using two connections with QUIC (similar to H2-Parallel) instead of emulating two connections using the congestion controller could improve QUIC performance in this scenario.

D. Fairness comparison

So far we have measured PLT of various protocols while running in isolation. Now we investigate their behavior while competing with one another. For this experiment we use two

clients connected to two servers using the same 7 Mbps bottleneck link. We host a synthetic web page with large JPG images on the web servers and both clients load the web page at the same time, but using a different protocol. H2 and QUIC use a single connection, H2-Parallel uses two connections while H1 uses four connections to load the page. We measure the throughput of each protocol using tcpdump. Figure 7 shows the bandwidth share of competing connections per protocol pair.

In Figure 7a we can see that QUIC gets twice as much bandwidth share as compared to H2 as it emulates two connections using its congestion controller. It has been shown in a recent study [14] that QUIC version 34 is unfair to TCP even when competing against 2 or even 4 TCP connections. However, we do not observe such behavior in our experiments with QUIC version 39. Figure 7b clearly shows that H2-Parallel using 2 TCP connections and QUIC version 39 get an equal share of bandwidth, as expected from QUIC's congestion controller. Figure 7c shows that H1 using 4 TCP connections is more aggressive than QUIC and thus has an unfair advantage when competing with both H2 and QUIC.

VI. Conclusion

We presented a performance evaluation of modern web protocols in adverse real-world scenarios. We confirmed that H2 exhibits suboptimal performance in such scenarios and suffers from unfairness when competing with other protocols due to its use of a single TCP connection. Results showed that QUIC is not as severely affected, because it implements a congestion controller that emulates the behavior of two TCP connections over UDP.

We implemented and evaluated a solution to improve H2 performance, called H2-Parallel, which lets browsers open multiple TCP connections for H2 as they usually do for H1. We compared H2-Parallel with QUIC, H1, H2 and H2-MP, which relies on MPTCP to open parallel subflows. H2-Parallel has interesting advantages: it presents performance similar to QUIC, profits from parallel Internet paths similar to H2-MP, and requires changes only in the client browser thus easing deployment. Our experiments show that using only two connections with H2-Parallel provides significantly better performance than regular H2 in the tested scenarios, hence it avoids overloading the network with large number of connections like H1.

References

[1] B. Thomas, R. Jurdak, and I. Atkinson, "SPDYing Up the Web," *Commun. ACM*, vol. 55, no. 12, pp. 64–73, 2012.

[2] M. Belsche, R. Peon, and M. Thomson, "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[3] A. Langley *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the SIGCOMM*, 2017, pp. 183–196.

[4] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the Web HTTP/2 Yet?" in *Proceedings of the PAM Conference*, 2016, pp. 218–232.

[5] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld, "How HTTP/2 Pushes the Web: An Empirical Study of HTTP/2 Server Push," in *Proceedings of the IFIP Networking Conference*, 2017.

[6] J. Manzoor, I. Drago, and R. Sadre, "How HTTP/2 is Changing Web Traffic and How to Detect It," in *Proceedings of the TMA Conference*, 2017.

[7] K. Zarifis, M. Holland, M. Jain, E. Katz-Bassett, and R. Govindan, "Modeling HTTP/2 Speed from HTTP/1 Traces," in *Proceedings of the PAM Conference*, 2016, pp. 233–247.

[8] H. de Saxcé, I. Oprescu, and Y. Chen, "Is HTTP/2 Really Faster than HTTP/1.1?" in *Proceedings of the IEEE Conference on Computer Communications Workshops*, 2015, pp. 293–299.

[9] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'ier Mobile Web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[10] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY Really Make the Web Faster?" in *Proceedings of the IFIP Networking Conference*, 2014, pp. 1–9.

[11] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *Proceedings of the NSDI*, 2014, pp. 387–399.

[12] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 609–614.

[13] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is quic?" in *Proceedings of the ICC*, 2016, pp. 1–6.

[14] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic," 2017.

[15] J. Manzoor, I. Drago, and R. Sadre, "The Curious Case of Parallel Connections in HTTP/2," in *Proceedings of the CNSM*, 2016, pp. 174–180.

[16] (2017) Chrome opening up to 6 connections with H2. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=718576

[17] B. Han, F. Qian, S. Hao, and L. Ji, "An Anatomy of Mobile Web Performance over Multipath TCP," in *Proceedings of the ACM CoNEXT*, 2015, pp. 5:1–5:7.

[18] B. Han, F. Qian, and L. Ji, "When Should We Surf the Mobile Web Using Both Wifi and Cellular?" in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (ATC)*, 2016, pp. 7–12.

[19] Cisco. (2017) BGP Best Path Selection Algorithm. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[20] Juniper. (2017) Understanding BGP Multipath. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/bgp-multipath-understanding.html

[21] B. Augustin, T. Friedman, and R. Teixeira, "Measuring Multipath Routing in the Internet," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 830–840, 2011.

[22] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002, pp. 97–105.

[23] Z. Cataltepe and P. Moghe, "Characterizing Nature and Location of Congestion on the Public Internet," in *Proceedings of the ISCC Symposium*, 2003, pp. 741–746.

[24] A. Akella, S. Seshan, and A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," in *Proceedings of the IMC*, 2003, pp. 101–114.

[25] A. Tachibana, A. Shigehiro, T. Hasegawa, M. Tsuru, and O. Yuji, "Locating Congested Segments over the Internet Based on Multiple End-To-End Path Measurements," *IEICE Transactions on Communications*, vol. 89, no. 4, pp. 1099–1109, 2006.

[26] J. Zhang, K. Xi, L. Zhang, and H. J. Chao, "Optimizing Network Performance using Weighted Multipath Routing," in *Proceedings of the ICCCN Conference*, 2012, pp. 1–7.

[27] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the Hotnets Workshop*, 2010, pp. 19:1–19:6.

[28] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband Internet Performance: A View from The Gateway," in *Proceedings of the SIGCOMM*, 2011, pp. 134–145.

[29] J. Sommers and P. Barford, "Cell vs. WiFi: On the Performance of Metro area Mobile Connections," in *Proceedings of the IMC*, 2012, pp. 301–314.

A Protocol-Ignorance Perspective on Incremental Deployability of Routing Protocols

Vadim Kirilin
IMDEA Networks Institute and UC3M, Spain

Sergey Gorinsky
IMDEA Networks Institute, Spain

***Abstract*—New protocols for Internet inter-domain routing struggle to get widely adopted. Because the Internet consists of more than 50,000 autonomous systems (ASes), deployment of a new routing protocol has to be incremental. In this work, we study such incremental deployment. We first formulate the routing problem in regard to a metric of routing cost. Then, the paper proposes and rigorously defines a statistical notion of protocol ignorance that quantifies the inability of a routing protocol to accurately determine routing prices with respect to the metric of interest. The proposed protocol-ignorance model of a routing protocol is fairly generic and can be applied to routing in both inter-domain and intra-domain settings, as well as to transportation and other types of networks. Our model of protocol deployment makes our study specific to Internet inter-domain routing. Through a combination of mathematical analysis and simulation, we demonstrate that the benefits from adopting a new inter-domain protocol accumulate smoothly during its incremental deployment. In particular, the simulation shows that decreasing the routing price by 25% requires between 43% and 53% of all nodes to adopt the new protocol. Our findings elucidate the deployment struggle of new inter-domain routing protocols and indicate that wide deployment of such a protocol necessitates involving a large number of relevant ASes into a coordinated effort to adopt the new protocol.**

I. Introduction

For a long time, Border Gateway Protocol (BGP) [1] has remained the only prominent protocol in inter-domain routing practice. Based on information propagated by neighboring nodes, a BGP node decides which paths it uses and which routing information it shares with neighbors. The local selection and filtering of path announcements by each node provides the node with means to realize its political, economic, and security policies.

On the flip side, BGP suffers from serious problems inherent in its design concept. The local filtering results in unintended information hiding, which artificially reduces the diversity of usable paths. BGP is vulnerable to hijacking [2]. Other problems include slow convergence and lack of support for multipath routing or end-to-end Quality of Service (QoS). While refinements of BGP mitigate some of its weaknesses [3]–[5], a thorough solution to the BGP problems requires changes in the conceptual design.

Radically different designs for inter-domain routing have been proposed and yet not deployed widely [6]–[8]. For example, Route Bazaar is a blockchain-inspired approach that uses a public ledger to announce, select, and verify end-to-end QoS-aware routing in a privacy-preserving manner [8]. Such solutions empower an autonomous system (AS) to not only enforce its security, political, and economic policies but also obtain flexible secure routing based on global information. Despite the promise of significant improvements, the new inter-domain protocols fail to get widely deployed.

ISBN 978-3-903176-08-9

This paper studies the problem of deploying a new inter-domain routing protocol. With the Internet composed by more than 50,000 ASes, replacement of BGP with a new protocol has to be incremental because it is virtually impossible for all the ASes to agree on simultaneously adopting the new protocol on a flag day. Moreover, the benefits of partially deploying the new protocol have to be significant compared to the extra hardware, staff training, and other expenses incurred by the protocol adopters. We develop a rigorous mathematical approach to understand how the extent of partial deployment affects the amount of the benefits realized by the deployment. The sought understanding is of practical importance due to its potential to both explain the deployment struggle of new inter-domain routing protocols and guide a successful deployment for such a protocol.

The cornerstone of our approach is a novel model of a routing protocol, which is equally applicable to inter-domain and intra-domain protocols. The model centers on the ability of a routing protocol to solve the routing problem with respect to a metric of routing cost. The metric of interest can be end-to-end path latency, monetary cost of traffic transit through other nodes, consumed network capacity, or a hybrid of multiple simple metrics. Given a global set of paths constrained by the political, economic, and security policies (if any) of the network nodes, the protocol determines prices of all network links with respect to the routing-cost metric and then constructs routing to deliver a global traffic-demand matrix along the available paths. The considered model of a routing protocol is fairly generic and can be applied to not only computer networking but also transportation and other kinds of networks.

The key innovation in our routing-protocol model is a statistical notion of *protocol ignorance* that quantifies the inability of a routing protocol to accurately determine the price of a network link with respect to the routing-cost metric. This inability arises due to various reasons:

1) A protocol is designed to operate with a different metric than the metric of interest. For example, if the metric of interest is latency, the routing prices of links are in general determined inaccurately by BGP, Routing

Information Protocol (RIP) [9], and the other routing protocols that use the hop count as a proxy for latency.
2) Even when the security, political, and economic policies of any node do not prohibit routing along a link, a protocol artificially excludes the link from the constructed paths, which effectively renders the link price infinite. For instance, this happens in BGP due to its local filtering and single-path routing.
3) Because the routing-cost metric changes its value over time, a protocol measures the dynamic value inaccurately. For example, when latency is the metric of interest, Open Shortest Path First (OSPF) [10] sets the link price to mean latency and ignores higher moments of latency, which still reduces the protocol ignorance of OSPF compared to RIP.

For simplicity, we model a link price as a random variable, rather than a stochastic process.

What makes our study specific to inter-domain routing protocols is our model of protocol deployment. With a *deployment trajectory* referring to a sequence of network nodes that adopt a new inter-domain routing protocol, we assume that each deployment trajectory for the same number of adopting nodes is equally likely in inter-domain settings because ASes act as independent players. This feature distinguishes our model of incrementally deploying an inter-domain routing protocol from intra-domain settings, where the operator of a domain can hand-pick adopting nodes and deployment trajectories to maximize the amount of benefits realized by incremental deployment of a new intra-domain routing protocol.

Via theoretical analysis and packet-level simulation, we evaluate how the routing cost changes with incremental deployment of a new inter-domain routing protocol. We analytically characterize the dependence of the routing cost on a change in routing. In its turn, the simulation examines how protocol ignorance affects routing when the new protocol is incrementally deployed. Combining the two dependencies reveals that the routing cost changes smoothly during incremental deployment. The main contributions of our paper are as follows:

- We propose and rigorously define a statistical notion of protocol ignorance that quantifies the inability of a routing protocol to accurately determine link prices with respect to a routing-cost metric. Our protocol-ignorance model of a routing protocol is equally applicable to inter-domain and intra-domain protocols.
- Based on the notion of a deployment trajectory, we model incremental deployment of a new inter-domain routing protocol.
- Our analysis and simulation show that the routing cost changes smoothly during incremental deployment of a new inter-domain protocol. This explains the struggle of new inter-domain routing protocols to get widely deployed and indicates that their successful deployment necessitates a coordinated adoption effort by a large number of relevant ASes.

Notation	Semantics
$G = (V, E)$	Network topology with node set V and edge set E
$g = \|V\|$	Number of nodes in topology G
$n = \|E\|$	Number of edges in topology G
Z	Set of source-destination pairs
$m = \|Z\|$	Number of source-destination pairs
$z = (s_z, d_z)$	Node pair with source s_z and destination d_z
$R = (r_z)^T$	Traffic demands of all source-destination pairs z
P_z	Set of available paths for source-destination pair z
$l_z = \|P_z\|$	Number of available paths for source-destination pair z
P	Set of all available paths in the topology
$l = \|P\| = \sum_{z=1}^{m} l_z$	Total number of available paths in the topology
p	Path
e	Edge
$B = (b_{ep})$	Edge composition of all available paths
$W_z = (w_{zp})^T$	Traffic-demand split for source-destination pair z
W	Traffic-demand splits for all source-destination pairs
U	All-ones vector
a_e	Aggregate traffic demand on edge e
$F(\cdot) = (f_e(\cdot))^T$	Routing-price functions on all edges
C_e	Routing cost on edge e
$C = \sum_{e \in E} C_e$	Total routing cost

TABLE I: Notation in our model of the routing problem.

The paper has the following structure. Section II presents our model. Sections III and IV evaluate the model via analysis and simulations respectively. Section V discusses related work. Finally, section VI sums up the paper and its contributions.

II. Modeling

A. Routing problem

While our model of a routing protocol focuses on its ability to solve a routing problem, we first formalize the routing problem. Table I sums up relevant notation. We model the network topology as a directed graph $G = (V, E)$ with $g = |V|$ nodes and $n = |E|$ edges. Set Z of size m contains all source-destination pairs $z = (s_z, d_z)$, which have traffic demands $R = (r_z)^T$. Set P_z of size l_z contains all paths available for source-destination pair z. Then, $P = \bigcup_{\forall z \in Z} P_z$ of size $l = \sum_{z=1}^{m} l_z$ constitutes the set of all available paths in the topology. Matrix $B = (b_{ep})$ of size $n \times l$ expresses the edge composition of all available paths. Bit b_{ep} is 1 for path p containing edge e and equals 0 otherwise.

The considered routing problem is a problem of splitting all traffic demands R among available paths. Our model is for multipath routing and includes single-path routing as its special case. With w_{zp} denoting the fraction of traffic demand r_z routed along path p, we express the split of this traffic demand as vector $W_z = (w_{zp})^T$. Constraint $W_z^T U = 1$ ensures routing for the entire demand of source-destination pair z, where U is an all-ones vector. To represent the traffic-demand splits of all source-destination pairs z, we compose block matrix W of size $l \times m$ by forming its diagonal from

Notation	Semantics
Ω	Traffic demand as a random variable
Λ	Edge price as a random variable
X	Ω or Λ
$\phi_X(t)$	Characteristic function of X
$\Phi_X(x)$	Cumulative distribution function of X
k	Moment order
μ_k	Estimate for the k-th lowest moment of X by a real protocol
q	Number of X's lowest moments estimated by a real protocol
$\rho_X(t)$	Estimated characteristic function of X
$\psi_e(\cdot)$	Estimated routing-price function on edge e
$\Psi(\cdot)$	Vector $(\psi_e(\cdot))^T$ of the estimated routing-price functions
α or β	Real routing protocol
$\mathfrak{i}_e^\alpha$	Protocol ignorance of protocol α on edge e
$\mathfrak{i}_e^{\alpha\beta}$	Relative protocol ignorance of protocols α and β on edge e
$\mathfrak{I}^{\alpha\beta}$	Relative protocol ignorance of protocols α and β

TABLE II: Notation in our model of a routing protocol.

vectors W_z and setting all its other elements to zero. Each row of matrix W corresponds to the same path as in the respective column of matrix B. Given the global traffic-demand splits, we add up the traffic demands on edge e to compute aggregate traffic demand a_e on each edge e.

We define the routing problem with respect to a metric of routing cost. Following the approach by Roughgarden and Tardos [11], our model determines routing cost C_e on edge e as the product of its traffic demand and routing price: $C_e = a_e f_e(a_e)$ where routing-price functions $F(\cdot) = (f_e(\cdot))^T$ on all edges are non-negative and monotonic. For example, the edge price can be a monetary price of transiting one Mbps of traffic along the edge, latency experienced by the traffic on the edge, or a combination of multiple simple metrics. The routing-price functions can represent such effects as limited edge capacities and congestion, e.g., account for congestion-induced latency when the price is latency. With $C = \sum_{e \in E} C_e$ denoting the total routing cost for the entire network, we formulate the routing problem as a minimization of this total cost:

$$\begin{aligned} &\text{minimize } C = F(BWR)^T BWR \\ &\text{under constraints } W_z^T U = 1 \ \forall z \in Z, \\ &\qquad\qquad w_{zp} \geq 0 \ \forall z \in Z \ \forall p \in P_z, \\ &\text{with inputs } G, Z, P, R, F(\cdot), \text{ and} \\ &\text{with outputs } W \text{ and } C. \end{aligned}$$

B. Routing protocol

While section II-A formalizes the routing problem, we now present our model of a routing protocol, with Table II reporting respective additional notation. Our protocol model abstracts away operational details of the protocol, such as the format of its control messages, events that trigger them, etc. Instead, we focus on the inability of a routing protocol to optimally solve the routing problem with respect to the metric of interest due to *protocol ignorance*, which refers to the inability of the protocol to accurately determine the routing prices of network links. This section introduces and rigorously defines the stochastic notion of protocol ignorance. Our protocol-ignorance model of a routing protocol is fairly general and applicable to not only inter-domain but also intra-domain routing, as well as to transportation and other types of networks.

This inability of a routing protocol to know the routing prices exactly arises due to a variety of reasons. First, the protocol might be designed to operate with a different metric than the metric of current interest. For instance, while BGP and RIP use the hop count as the metric of routing cost, the metric of current interest might be latency, and the prominent hop-based protocols determine the routing prices of network links in regard to the latter metric imprecisely. Furthermore, the hop count is increasingly becoming a less representative proxy for path latency due to massive emergence of tunneling techniques that make some hops invisible to the routing protocol, e.g., because of remote peering in Internet inter-domain routing [12].

Second, the design of a routing protocol might unnecessarily exclude a link from routing some traffic, which effectively renders the link price infinite for the purposes of routing this traffic. For example, such link exclusion occurs in BGP due to local filtering of a path by an AS even when routing along the excluded link does not violate any economic, political, or security policy of any AS. Also, single-path routing in BGP unnecessarily prevents routing of some traffic along some links, which similarly undermines the ability of BGP to solve the routing problem optimally. Note that although single-path routing and local filtering in BGP simplify the protocol design and improve its scalability, these design choices are not fundamental for Internet inter-domain routing. For instance, Route Bazaar is an alternative inter-domain routing approach that supports multipath routing and uses a decentralized global public ledger for enabling each AS to make local routing decisions and enforcing the security, political, and economic policies of all ASes in a privacy-preserving manner.

Third, even when a protocol is designed for the same routing metric of interest, the protocol might be unable to exactly measure the dynamic values of the metric. For example, the values of path latency continuously change due to packet queuing in network nodes.

Regardless of the reasons why a particular protocol does not know the exact routing prices, the statistical notion of protocol ignorance quantifies this inability. Below, we refer to a protocol with imperfect knowledge of the routing prices as a *real protocol*. An *optimal protocol* measures the routing prices exactly.

1) Representation of an optimal protocol: For each edge e, we view its aggregate traffic demand and routing price as random variables Ω and Λ respectively and refer to either of them as X for exposition brevity. The characteristic function of X is $\phi_X(t) = \mathbb{E}[e^{itX}]$ where i is the imaginary unit, and $t \in \mathbb{R}$. In our model, an optimal protocol knows exactly all moments $\mathbb{E}[X^k]$ of X, where $k = 1, 2, \ldots, \infty$. According to the Hausdorff moment problem [13], the collection of all the moments uniquely determines the probability density function

(PDF) of X. Specifically, assuming that $\phi_X(t)$ is an analytic function, the optimal protocol expands it into a Taylor series:

$$\phi_X(t) = 1 + \sum_{k=1}^{\infty} \frac{(it)^k}{k!}\mathbb{E}[X^k] \tag{1}$$

and recovers the PDF of X from $\phi_X(t)$ through the inverse Fourier transform as $\frac{1}{2\pi}\int_{\mathbb{R}} \phi_X(t)e^{itx}dt$. By integrating the obtained PDFs of Ω and Λ, the optimal protocol obtains the cumulative distribution function (CDF) for each of these two random variables, $\Phi_\Omega(x)$ and $\Phi_\Lambda(x)$ respectively. Because routing-price function $f_e(\cdot)$ is monotonic, the optimal protocol computes it as $f_e(\cdot) = \Phi_\Omega^{-1}(\Phi_\Lambda(x))$ for each edge e and solves the routing problem of section II-A optimally.

2) Representation of a real protocol: On the other hand, a real protocol observes only samples drawn from the probability distribution of variable X and uses them to compute estimates μ_k for the q lowest moments of X, i.e., for $k = 1, \ldots, q$. The real protocol computes an estimated characteristic function $\rho_X(t)$, an estimate of $\phi_X(t)$, as:

$$\rho_X(t) = 1 + \sum_{k=1}^{q} \frac{(it)^k}{k!}\mu_k. \tag{2}$$

By applying the inverse Fourier transform to $\rho_X(t)$ and then integrating the obtained PDF, the real protocol computes estimated routing-price functions $\Psi(\cdot) = (\psi_e(\cdot))^T$ and uses them instead of functions $F(\cdot) = (f_e(\cdot))^T$ when solving the routing problem of section II-A. Because $\Psi(\cdot)$ are only estimates of $F(\cdot)$, the real protocol computes routing W and its total cost C suboptimally in general.

3) Relevance to prominent existing protocols: Whereas existing routing protocols do not actually perform inverse Fourier transforms, integration, or other complicated operations described above, we now show that our model of a routing protocol realistically represents the handling of routing prices by prominent existing protocols.

Hop-based protocols. This kind of routing protocols uses the hop count as the metric of routing cost. BGP and RIP are prominent representatives of such protocols in the inter-domain and intra-domain settings respectively. In our model, a hop-based protocol does not measure any moments of X, i.e., $\mu_k = 0$ for $k = 1, \ldots, \infty$, even when the routing metric of interest has dynamic values, e.g., when the metric of interest is latency. Thus, the respective estimated characteristic function is $\rho_X(t) = 1$. The inverse Fourier transform produces the Dirac delta function as the PDF of X, implying that X is a constant and that the edge cost is the same for all the edges, i.e., the model realistically represents the link pricing in a hop-based protocol.

Mean-measuring protocols. A mean-measuring protocol measures only the first moment, i.e., mean μ_1, of routing price Λ. OSPF is a prominent mean-measuring intra-domain protocol when it is configured to measure the routing price as mean latency, e.g., by using a sliding window estimation. While the hop-based BGP constitutes the only prominent existing protocol for inter-domain routing, Route Bazaar is an alternative Internet connectivity approach where mean-measuring protocols can be used for inter-domain routing. In our model of a mean-measuring protocol, the corresponding estimated characteristic function is $\rho_\Lambda(t) = 1 + (it)^k\mu_1$. The inverse Fourier transform yields $H(x) - \mu_1\delta'(x)$ as the PDF of Λ, where $H(x)$ denotes the Heaviside step function, and $\delta'(x)$ is the derivative of the Dirac delta function. The integration of this function leads to estimating each edge cost as the mean of the metric, i.e., the model realistically represents the handling of routing prices by a mean-measuring protocol.

4) Mathematical definition of protocol ignorance: To model how accurately a real protocol α estimates edge price Λ in comparison to an optimal protocol, we define *protocol ignorance* $\mathfrak{i}_e^\alpha$ of protocol α on edge e as:

$$\mathfrak{i}_e^\alpha = \int_0^c \Big|\rho_\Lambda(t) - \phi_\Lambda(t)\Big|dt \tag{3}$$

where c is a constant ensuring existence of the integral. Based on equations 1 and 2, we express this protocol ignorance as:

$$\mathfrak{i}_e^\alpha = \int_0^c \Big|\sum_{k=0}^{\infty}(\frac{(it)^k}{k!}(\mathbb{E}[\Lambda^k] - \mu_k)\Big|dt \tag{4}$$

where $\mu_k = 0$ for $k > q$. The protocol ignorance of an optimal protocol equals 0. For a real protocol α, we have $\mathfrak{i}_e^\alpha > 0$. As the real protocol estimates more moments of Λ and measures each moment more accurately, $\mathfrak{i}_e^\alpha$ decreases toward 0, and the smaller protocol ignorance enables real protocol α to estimate the routing-price function on edge e more accurately.

The notion of protocol ignorance forms a basis for comparing two real routing protocols α and β. We define *relative protocol ignorance* of protocols α and β on edge e as:

$$\mathfrak{i}_e^{\alpha\beta} = \lim_{c\to\infty} \frac{\mathfrak{i}_e^\alpha - \mathfrak{i}_e^\beta}{c^{1+\max\{q^\alpha, q^\beta\}}} \tag{5}$$

which no longer depends on the choice of constant c. Here, q^α and q^β denote the number of moments estimated for edge price Λ by protocols α and β respectively. Vector of $\mathfrak{i}_e^{\alpha\beta}$ for all edges e in E provides a topology-wide perspective on the relative protocol ignorance. We define *relative protocol ignorance* of protocols α and β as a norm of this vector:

$$\mathfrak{I}^{\alpha\beta} = \sqrt{\sum_{e\in E}\left(\mathfrak{i}_e^{\alpha\beta}\right)^2}. \tag{6}$$

Example 1. Let α and β refer respectively to hop-based and mean-measuring protocols. For edge e, protocol β observes the following five samples of edge latency Λ, which has an exponential distribution: 0.81, 0.63, 2.10, 1.02, and 0.66. Using the samples, protocol β computes $\mu_1 = 1.044$ as an estimate of moment $\mathbb{E}[\Lambda]$, and $\rho_\Lambda(t) = \frac{1}{1-it}$ as an estimate of characteristic function $\phi_\Lambda(t)$. Then, the protocol ignorance of protocol β on edge e is $\mathfrak{i}_e^\beta = |\ln(1-ic) - c - 0.522ic^2|$. Protocol α, which does not measure the edge latency at all, has a larger protocol ignorance $\mathfrak{i}_e^\alpha = |\ln(1-ic)|$. Thus, the relative protocol ignorance of protocols α and β on edge e is $\mathfrak{i}_e^{\alpha\beta} = 0.022$, confirming the better awareness of protocol β

Notation	Semantics
h	Number of nodes that adopt the new protocol
j	Deployment trajectory
$\Psi_{hj}(\cdot)$	Routing-price functions for deployment trajectory j of h nodes
W_{hj}	Routing for deployment trajectory j of h nodes
C_{hj}	Routing cost for deployment trajectory j of h nodes
C_h	Average routing cost C_h for all deployments of h nodes
Θ	Edge-sharing matrix
γ^{ab}	Bilinear form
L_e	Lipschitz constant of estimated price function $\psi_e(\cdot)$ on edge e
L	Maximum L_e among all edges e
Υ_z^a	Auxiliary block matrix in the proof of theorem 2

TABLE III: Notation in our model of protocol deployment.

about the edge latency. If protocol β estimated the both lowest moments of edge latency Λ, its protocol ignorance on edge e would change to $\mathfrak{i}_e^\beta = |ln(1-ic)-c-0.522ic^2+0.463c^3|$, and the relative protocol ignorance of protocols α and β on this edge would increase to $\mathfrak{i}_e^{\alpha\beta} = 0.13$, representing the increased advantage of protocol β over protocol α in knowing the latency distribution on edge e. $\triangle$

C. Incremental deployment of a new inter-domain protocol

While the model of a routing protocol in section II-B is equally applicable to inter-domain and intra-domain protocols, we now present our model for incremental protocol deployment specific to inter-domain routing protocols. Table III reports corresponding extra notation.

Suppose that all nodes in topology G support an incumbent inter-domain routing protocol α. Adopting a new inter-domain routing protocol β can reduce the routing cost in topology G because protocol β measures more accurately the routing prices on those edges where the new protocol is used. Some nodes deploy protocol β. When a node deploys protocol β, this protocol is used on all outcoming edges of this node. Protocol β is backward compatible with protocol α and runs on top of the incumbent protocol, e.g., by using Generic Route Encapsulation (GRE) tunnels [14] or another tunneling technique.

The routing and its cost depend on not only how many nodes adopt the new protocol but also which specific nodes are the adopters. Hence, we define a *deployment trajectory* of h nodes as a sequence of the first h adopting nodes in the order of their deployment of protocol β. Because ASes in the practice of inter-domain routing act as independent players, we assume that every deployment trajectory of h nodes is equally likely. The equal likelihood of deployment trajectories is the main feature distinguishing our inter-domain deployment model from intra-domain settings, where the domain operator can cherry-pick h adopting nodes to maximize the reduction in the routing cost. For the inter-domain settings, we express average routing cost C_h for all deployments of h nodes as:

$$C_h = \frac{1}{P(g,h)} \sum_{j=1}^{P(g,h)} C_{hj} \tag{7}$$

where g is the number of nodes in the topology, C_{hj} refers to the routing cost for the deployment of h nodes that has the j-th h-permutation of g as its trajectory, and $P(|V|,h)$ is the total number of such h-permutations of g.

Let us examine a full deployment of g nodes with trajectory j. Estimated routing-price functions $\Psi_{hj}(\cdot)$, routing W_{hj}, and routing cost C_{hj} for a deployment of h nodes might all change at each stage h along this trajectory, where $h = 1,\ldots,g$. As h increases, cost C_{hj} changes due to two conflated effects: (a) changes in routing W_{hj} and (b) changes in estimates $\Psi_{hj}(\cdot)$ of routing-price functions $F(\cdot)$. To segregate the two effects, we can track the value of $\tilde{C}_{hj} - C_{gj}$ at each stage h, where C_{gj} is the routing cost with the full deployment of protocol β, and $\tilde{C}_{hj}$ denotes the cost of routing W_{hj} computed with full-deployment routing-price functions $\Psi_{gj}(\cdot)$.

III. Analysis

The salient outcomes of our extensive modeling effort in section II include the formulation of the routing problem with respect to a metric of routing cost, statistical notion of protocol ignorance that quantitatively characterizes the inability of a protocol to measure routing prices accurately, and model for incremental deployment of a new inter-domain routing protocol. This section analyzes such incremental deployment. Specifically, we assess how much a change in routing affects the routing cost. The analysis is the first step towards understanding why BGP remains the only prominent inter-domain routing protocol and what fraction of the Internet ASes need to adopt a new inter-domain routing protocol to substantially benefit from the adoption.

A. Routing for one source-destination pair

For ease of exposition, we start the analysis by considering the simple scenario where the routing problem needs to be solved for only one source-destination pair z_1, i.e., $Z = \{z_1\}$. The set of available paths for the pair is P_1. Without loss of generality, we normalize the traffic demand of pair z_1 to $r_1 = 1$. Protocol β computes estimates $\Psi(\cdot)$ of routing-price functions $F(\cdot)$ as described in section II-B. With this, protocol β solves the following instance of the routing problem from section II-A:

$$\begin{aligned}
&\text{minimize } C = \Psi(BW_1)^T BW_1 \\
&\text{under constraints } W_1^T U = 1, \\
&\qquad\qquad w_{1p} \geq 0 \ \forall p \in P_1, \\
&\text{with inputs } G, \{z_1\}, P_1, r_1 = 1, \Psi(\cdot), \text{ and} \\
&\text{with outputs } W_1 \text{ and } C.
\end{aligned}$$

Consider two routings W_1^x and W_1^y that have costs C^x and C^y respectively. Vector $W_1^\epsilon = W_1^x - W_1^y$ of size $l_1 = |P_1|$ represents the difference between these routings. Let $\Theta = (\theta_{pu})$ of size $l_1 \times l_1$ denote an edge-sharing matrix $B^T B$ (where matrix B expresses the edge composition of all available paths), and θ_{pu} represents the number of edges shared by paths p and u, implying that θ_{pu} is at most the diameter of topology G. Then, we represent bilinear form $(W_1^a)^T \Theta W_1^b$

as γ^{ab}. By construction, estimated routing-price functions $\Psi(\cdot)$ have Lipschitz continuity. We define constant $L = \max_{e \in E}\{L_e\}$ where L_e is the Lipschitz constant of estimated routing-price function $\psi_e(\cdot)$ on edge e.

Theorem 1. *In routing for one source-destination pair, a change in the total routing cost is bounded from above as follows:*

$$|C^x - C^y| \leq L(|\gamma^{\epsilon\epsilon}| + 2|\gamma^{\epsilon y}|). \quad (8)$$

Proof. Let C^{ab} denote $\Psi(BW_1^a)^T BW_1^b$. Then, we express cost C^x as $\Psi(BW_1^x)^T B(W_1^y + (W_1^x - W_1^y)) = \Psi(BW_1^x)^T BW_1^y + \Psi(BW_1^x)^T W_1^\epsilon = C^{xy} + C^{x\epsilon}$. Similarly, we express cost C^y as $\Psi(BW_1^y)^T B(W_1^x - (W_1^x - W_1^y)) = \Psi(BW_1^y)^T BW_1^x - \Psi(BW_1^y)^T W_1^\epsilon = C^{yx} - C^{y\epsilon}$. Thus, we have:

$$C^x = C^{xy} + C^{x\epsilon} \text{ and } C^y = C^{yx} - C^{y\epsilon} \quad (9)$$

and express the sum of these two costs as:

$$C^x + C^y = C^{xy} + C^{yx} + C^{x\epsilon} - C^{y\epsilon}. \quad (10)$$

Using equation 10, we express the difference of the two costs as:

$$C^x - C^y = (C^{xy} - C^y) + (C^{yx} - C^y) + (C^{x\epsilon} - C^{y\epsilon}). \quad (11)$$

The three terms on the right-hand side of equation 11 have the following upper bounds: $|C^{xy} - C^y| = |(\Psi(BW_1^x)^T - \Psi(BW_1^y)^T)BW_1^y| \leq L|\epsilon^T BW_1^y| = L|\gamma^{\epsilon y}|$, $|C^{yx} - C^y| = |\Psi(BW_1^y)^T BW_1^\epsilon| = |(\Psi(BW_1^y)^T - \Psi(BO)^T)BW_1^\epsilon| \leq L|\gamma^{y\epsilon}|$, and $|C^{x\epsilon} - C^{y\epsilon}| = |(\Psi(BW_1^x)^T - \Psi(BW_1^y)^T)BW_1^\epsilon| \leq L|\gamma^{\epsilon\epsilon}|$ where O is a zero vector. Because the symmetry of matrix Θ implies $\gamma^{y\epsilon} = \gamma^{\epsilon y}$, we combine the above three bounds to derive equation 8. □

B. Routing for an arbitrary set of source-destination pairs

Now, we extend the result of theorem 1 for the general formulation of the routing problem in section II-A, i.e., when set Z of source-destination pairs and traffic demands R are arbitrary. The extension is fairly straightforward and largely related to generalizing the notation from vectors to matrices. In particular, matrix W^ϵ denotes the difference between two routings W^x and W^y, where W_z^ϵ equals $W_z^x - W_z^y$. Also, we define $\gamma_z^{\epsilon\epsilon} = (W_z^\epsilon)^T \Theta_z (W_z^\epsilon)$ where Θ_z equals $B_z^T B_z$.

Theorem 2. *In the general routing problem, a change in the total routing cost is bounded from above as follows:*

$$|C^x - C^y| \leq L(|\sum_{z=1}^{m} r_z^2 \gamma_z^{\epsilon\epsilon}| + 2|\sum_{z=1}^{m} r_z^2 \gamma_z^{\epsilon y}|). \quad (12)$$

Proof. By substituting W_1^x and W_1^y with $W^x R$ and $W^y R$ respectively, we follow the reasoning pattern in the proof of theorem 1 to show that

$$|C^x - C^y| \leq L(|(W^\epsilon R)^T \Theta (W^\epsilon R)| + 2|(W^\epsilon R)^T \Theta (W^y R)|).$$

Let Υ_z^a denote an auxiliary block matrix that has the same size as W. Its z-th block is W_z^a, and all the other elements equal zero. Then, we express W^a as $\sum_{z=1}^{m} \Upsilon_z^a$. Because $(\Upsilon_z^a R)^T \Theta (\Upsilon_j^b R)$ is zero for $j \neq z$, we represent $(W^a R)^T \Theta (W^b R)$ as $\sum_{z=1}^{m} (\Upsilon_z^a R)^T \Theta (\Upsilon_z^b R)$. By expressing $(\Upsilon_z^a R)^T \Theta (\Upsilon_z^b R)$ as $r_z (W_z^a)^T \Theta_z (W_z^b)$, we derive:

$$(W^a R)^T \Theta (W^b R) = \sum_{z=1}^{m} r_z (W_z^a)^T \Theta_z (W_z^b). \quad (13)$$

By applying equation 13 to both terms on the right-hand side of the bound earlier in the proof, we establish equation 12. □

The above results demonstrate that a change in the routing affects the routing cost smoothly, i.e., a small change in routing do not cause a large change in the routing cost.

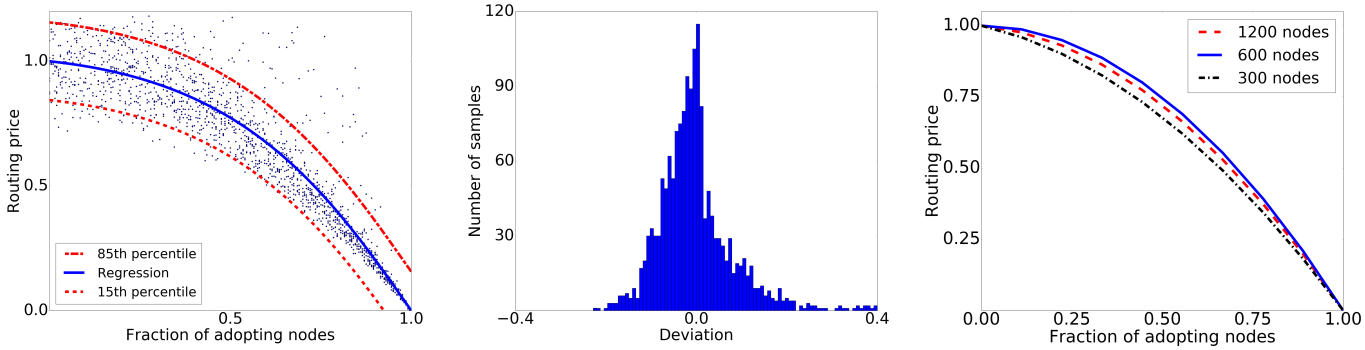
IV. Simulation

The analytic results in section III tell only half the story. They show how a change in routing affects the routing cost. To complete the story, we now examine how the lower ignorance of an incrementally deployed inter-domain protocol affects routing.

A. Methodology

We use real network data to conduct packet-level simulation. Simulating the global Internet faces two steep challenges: scale and fidelity. Because the Internet consists of more than 50,000 ASes connected by around a million inter-domain links, simulation of the entire topology would require enormous computational resources. Furthermore, neither the Internet topology nor its traffic-demand matrix is known with high precision for such simulation to produce highly accurate quantitative answers. In dealing with these challenges, we openly admit necessary limitations of the simulated model (such as using a single node to represent an AS), avoid a focus on exact quantitative results, and instead strive to expose the qualitative dependence of routing on protocol ignorance.

1) Topology: To tackle the challenge of topology scale, we characterize statistical properties of the global Internet topology and generate a family of smaller topologies with the same statistical properties. Specifically, we reconstruct snapshots of the AS-level Internet topology from the CAIDA dataset [15] based on traceroute [16] measurements. Our characterization of the snapshots confirms the observation that the AS-level Internet topology is a scale-free graph. Route Views [17] and other prominent sources of Internet connectivity data can be used to make the same observation. Based on the scale-free characterization, we use NetworkX [18], [19] to generate synthetic topologies that preserve the statistical properties of the global Internet topology and range in their size from 100 to 1,200 nodes, with the default size of 500 nodes. The probability to add an edge to a node during the topology generation varies between 0 and 50%, with 10% being the default value. The minimum number of edges adjacent to each node equals 3 by default and changes from 1 to 50.



(a) 2,000 runs with 10 runs per setting (b) Deviations of the 2,000 samples (c) Regression for different topology sizes

Fig. 1: Impact of protocol ignorance on the routing price during incremental deployment of the new inter-domain protocol.

2) Traffic matrix: While the Transmission Control Protocol (TCP) [20] transmits packets in bursts, the basic level of traffic in the simulation is a packet burst. Each source-destination pair communicates around 10,000 packet bursts. We size every packet to 64 KB. The number of packets in a burst varies from 1 to 32 and is distributed binomially with 32 independent experiments and success probability 0.5. Time between subsequent packet bursts has a random distribution with a mean of 1 ms. We consider three such distributions: exponential, uniform, and Weibull, with the exponential distribution being used in the default settings. We include the Weibull distribution due to prior measurement studies [21], [22] and independently validate it on another CAIDA dataset [23].

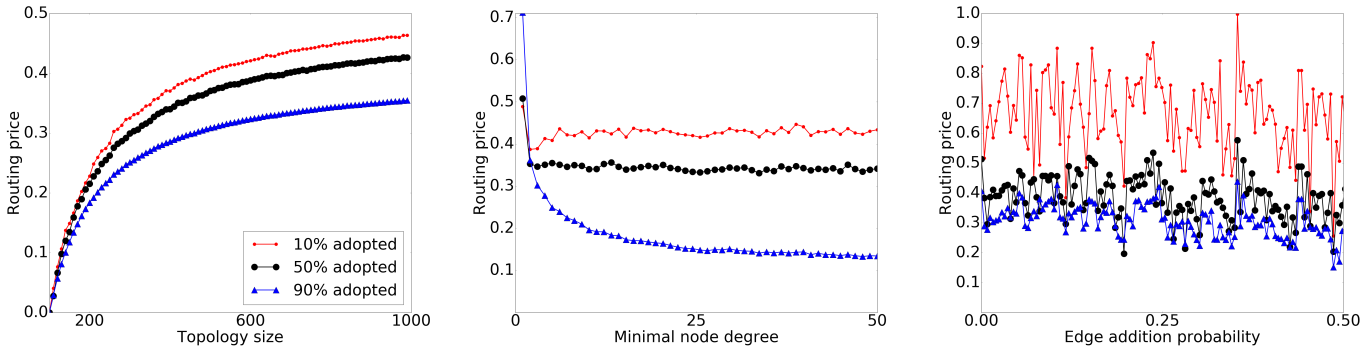
3) Routing protocols: The routing metric of interest in the simulation is latency. For the incumbent and new inter-domain routing protocols, we respectively consider the hop-based and mean-measuring protocols described in section II-B.

4) Simulator: To improve scalability of the simulation, we develop and utilize our own simulator. Unlike ns-3 [24] and other generic simulators that support many features at the price of significant overhead, our tool is customized for the problem in hand to scalably simulate traffic generation, routing, and delivery for each source-destination pair in the network topology. The tool is a discrete-time event simulator that represents each AS as a single node. In addition to generating packet bursts, every node also forwards packet bursts from other sources. The node forwards all packets of a burst together as a whole. The forwarded burst experiences transmission latency determined by dividing the burst size by the internal capacity of the AS; this internal capacity of the node is drawn from a truncated normal distribution. Additionally, the forwarded packet burst experiences queuing latency drawn from the exponential distribution with a mean of 0.05 ms. The simulator keeps the average network utilization at 50% by: (1) setting the capacity of each edge according to closeness centrality of both nodes incident to this edge and (2) then characterizing each edge with extra latency drawn from the same exponential distribution for all edges, with the rate parameter of this distribution being determined experimentally. For every simulated setting, we conduct 10 runs and, for each run, measure the routing price as the average end-to-end latency in the network. The code of our simulator is available in [25].

B. Simulation results

1) Impact of protocol ignorance: Figure 1a depicts how the routing price changes when the fraction of nodes adopting the new protocol increases with a step of 0.5% from 0 to 100%, i.e., from no deployment to full deployment. For each of the 10 runs in every simulated setting, we plot the routing price as a point. Figure 1a also plots a polynomial regression and its 15th and 85th percentiles for the results, with the elbow method consistently identifying a quadratic regression as the best fit. We normalize the plotted results by linearly scaling them to map interval $[f,n]$ into [0,1] where f and n refer to the regression values in the full-deployment and no-deployment settings respectively. Figure 1b reports a histogram of the deviations of the individual results from the corresponding regression values. Figure 1c plots the regression for three other sizes of the network topology. The dependence of the routing price on the deployment extent exhibits the same qualitative profile and only minor quantitative differences. The routing price undergoes a smooth quadratic decline over the entire range of incremental deployment. For the four considered sizes of the topology, between 43% and 53% of all nodes have to adopt the new protocol to decrease the routing price by 25%.

Combining the simulation insights with the analytical results from section III, we conclude that the benefits from adopting the new inter-domain protocol accumulate smoothly during incremental deployment and that protocol deployment by natural early adopters [26], [27] is insufficient to incentivize other ASes to deploy the protocol later. Our findings explain the struggle of new Internet inter-domain routing protocols to get widely deployed. Our results also indicate that widespread deployment of a new inter-domain protocol necessitates involving a large number of relevant ASes into a coordinated effort to adopt the protocol.



(a) Topology size (b) Minimal connectivity (c) Edge addition probability

Fig. 2: Sensitivity of the routing price to parameters of the topology generation algorithm.

2) Parameter sensitivity: Figure 1c already shows that the topology size makes a small quantitative impact on the quadratic dependence between the deployment extent and routing price. We also evaluate sensitivity of this dependence to the distribution of inter-burst time (exponential, uniform, or Weibull) and parameters of the topology generation algorithm. Compared to the results in figure 1c, these sensitivity studies unveil the same quadratic qualitative profile and even smaller quantitative differences for the dependence of the routing price on the fraction of adopting nodes. Due to such similarity and space constraints, we do not report the respective graphs here.

Figure 2 explores sensitivity of the routing price to three parameters of the topology generation algorithm when the fraction of adopting nodes is fixed at 10%, 50%, or 90%. We normalize the plotted results by linearly scaling them to map interval [l,h] into [0,1] where l and h refer respectively to the minimum and maximum routing price across all settings in these three parameter sensitivity studies.

Figure 2a shows that the routing price grows sublinearly as the topology size increases from 100 to 1,000 nodes. The result corroborates the intuition that the topology diameter grows slower than the topology size. For scale-free graphs, [28] analytically shows that the diameter grows on average with rate $\frac{log(g)}{log(log(g))}$ where g is the number of nodes. Because the diameter is determined by the longest shortest path, and the end-to-end routing cost grows on average linearly with the path length, the dependency depicted in figure 2a matches the theoretical expectation.

Figure 2b exhibits dependence of the routing price on the minimal node degree in the topology. With a larger fraction of adopting nodes, the routing price falls steeper as the minimal node degree increases from 1 to 50. This happens due to dependency between the node degree and number of paths in the topology. Because the incumbent protocol uses the number of hops as a proxy metric for latency and thus estimates actual routing prices inaccurately, the decrease in the routing price is more pronounced for larger deployments of the new protocol.

Figure 2c reports on varying the probability of adding a random edge during the topology construction. The topology generation algorithm keeps the number of added edges proportional to the topology size. Whereas the increase of the minimal node degree in our previous sensitivity study weakens the scale-free property of the topology, the addition of random edges strengthens this property without increasing the number of paths exponentially. Figure 2c demonstrates low sensitivity of the routing price to the edge addition probability.

Overall, among all conducted sensitivity studies, the routing price is most sensitive to the topology size and fraction of adopting nodes.

V. Related work

While prior work on inter-domain routing is extensive, its main focuses are not on the problem of incrementally transitioning from BGP to a new protocol. Even those papers that explicitly consider incremental migration to the new protocol tend to deal with technical issues of the transition [6] and do not provide clear answers on economic incentives for adopting ASes, especially for early adopters [26], [27]. Whereas [29] proposes a method for service composition that can be used to combine different routing protocols, the paper develops the marketplace support without studying incremental adoption of the marketplaces. [30] argues that it is possible to select a relatively small set of routing brokers, about 7% of the Internet ASes, to enable QoS-aware and, in particular, latency-aware routing for most of the Internet; while [30] hand-picks the routing brokers among strategically positioned ASes, our work makes a more realistic assumption that ASes adopt the new routing protocol voluntarily and randomly.

Incremental deployability attracts more direct attention in other problem domains of computer networking. In the context of Internet addressing, [31] studies incremental migration from IPv4 to IPv6 and estimates its costs. [32] analyzes a potential way to incrementally deploy a secure version of BGP. Our paper differs from these previous efforts in not only tackling a different problem domain but also using a new method based on protocol ignorance. The distinguishing trait of our work is its model that captures the inability of a protocol to estimate costs accurately.

Our work leverages various prominent previous efforts. The protocols designed in [6], [8], [33]–[35] inspire us to develop the concept of protocol ignorance. Our analysis extends the classical theoretical work by Roughgarden and Tardos on the price of anarchy in different types of networks [11]. Their research paves the way to formalize the routing problem and characterize dependence of the routing cost on routing. [11], [36] analyze different scenarios of network behaviour in its dependency on node behaviour. Our simulations rely on realistic network topologies [37] and use real traffic traces collected by CAIDA [15], [23]. The approaches in [21], [22], [38] guide our modeling work.

[39], [40] report on mathematical modeling of Internet protocols. Our work belongs to the same type of research. We develop a novel abstract model for inter-domain routing that ties together a routing protocol, routing constructed by the protocol, and cost of the constructed routing.

VI. Conclusion

In this paper, we studied incremental deployment of a new inter-domain routing protocol in the Internet. The paper formalized the routing problem in terms of minimizing a metric of routing cost. Then, we introduced and rigorously defined a statistical notion of protocol ignorance that quantifies the inability of a routing protocol to accurately determine routing prices with respect to the metric of interest. Our protocol-ignorance model of a routing protocol is fairly generic and applicable to not only inter-domain but also intra-domain routing, as well as to transportation and other kinds of networks. The considered model of protocol deployment made our study specific to Internet inter-domain routing. Using theoretical analysis and simulation, we showed that the benefits from adopting the new inter-domain protocol accumulate smoothly during incremental deployment. In the simulated topologies, between 43% and 53% of all nodes had to adopt the new protocol to decrease the routing price by 25%. Our results explained the lack of widespread adoption for new inter-domain routing protocols and indicated that their successful deployment necessitated a coordinated adoption effort by a large number of relevant ASes.

VII. Acknowledgments

This research was financially supported in part by the Regional Government of Madrid on Cloud4BigData grant S2013/ICE-2894.

References

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, 2006.
[2] P. Bangera and S. Gorinsky, "Impact of Prefix Hijacking on Payments of Providers." COMSNETS, 2011.
[3] J. Karlin, S. Forrest, and J. Rexford, "Pretty Good BGP: Improving BGP by Cautiously Adopting Routes." ICNP, 2006.
[4] M. Caesar and J. Rexford, "BPG Routing Policies in ISP Networks," *IEEE Network, 19(6)*, 2005.
[5] W. Sun, Z. Mao, and K. Shin, "Differentiated BGP Update Processing for Improved Routing Convergence." ICNP, 2006.
[6] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet Routing." SIGCOMM, 2009.
[7] V. Valancius, N. Feamster, R. Johari, and V. Vazirani, "MINT: A Market for INternet Transit." ReArch, 2008.
[8] I. Castro, A. Panda, B. Raghavan, S. Shenker, and S. Gorinsky, "Route Bazaar: Automatic Interdomain Contract Negotiation." HotOS, 2015.
[9] C. Hedrick, "Routing Information Protocol," RFC 1058, 1998.
[10] J. Moy, "OSPF Version 2," RFC 2328, 1998.
[11] T. Roughgarden and E. Tardos, "How Bad is Selfish Routing?" *Journal of the ACM, 49(2)*, 2002.
[12] I. Castro, J. C. Cardona, S. Gorinsky, and P. Francois, "Remote Peering: More Peering Without Internet Flattening." CoNEXT, 2014.
[13] J. Shohat and J. Tamarkin, *The Problem of Moments*. American Mathematical Society, 1943.
[14] D. Farinacci, S. Hanks, and P. Traina, "Generic Routing Encapsulation (GRE)," RFC 1701, 1994.
[15] CAIDA, "Anonymized Internet Traces 2012," 2016. [Online]. Available: http://www.caida.org/data/request_user_info_forms/ark.xml
[16] V. Jacobson, "Traceroute," 1989.
[17] University of Oregon, "Routeviews Prefix to AS Mappings Dataset (pfx2as) for IPv4 and IPv6," 2005. [Online]. Available: http://www.routeviews.org/routeviews/
[18] A. Hagberg, D. Schult, and P. Swar, "Exploring Network Structure, Dynamics, and Function using NetworkX." SciPy, 2008.
[19] P. Holme and B. Kim, "Growing Scale-Free Networks with Tunable Clustering," *Physical review E, 65(3)*, 2002.
[20] Information Sciences Institute University of Southern California, "Transmission Control Protocol," RFC 793, 1981.
[21] N. Hariri, B. Hariri, and S. Shirmohammadi, "A Distributed Measurement Scheme for Internet Latency Estimation," *IEEE Transactions on Instrumentation and Measurement, 60(5)*, 2011.
[22] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King : Estimating Latency between Arbitrary Internet End Hosts." IMW, 2002.
[23] CAIDA, "IPv4 Routed /24 Topology Dataset," 2016. [Online]. Available: http://www.caida.org/data/passive/passive_dataset_request.xml
[24] NS-3 Consortium, "ns-3," 2017. [Online]. Available: https://www.nsnam.org/
[25] V. Kirilin and S. Gorinsky, "Simulator Source Code," 2017. [Online]. Available: https://github.com/WVadim/SimulatorC-Large
[26] C. Catalini and C. Tucker, "When Early Adopters Don't Adopt," *Science, 365(6)*, 2017.
[27] J. R. Douceur and T. Moscibroda, "Lottery Trees: Motivational Deployment of Networked Systems." SIGCOMM, 2007.
[28] B. Bollobás and O. Riordan, "The Diameter of a Scale-Free Random Graph," *Combinatorica, 24(1)*, 2004.
[29] S. Bhat, R. Udechukwu, R. Dutta, and G. N. Rouskas, "On Service Composition Algorithm for Open Marketplaces of Network Services." EuCNC, 2017.
[30] D. Lin, D. Hui, W. Wu, T. Liu, Y. Yang, Y. Wang, J. C. Lui, G. Zhang, and Y. Li, "On the Feasibility of Inter-Domain Routing via a Small Broker Set." ICDCS, 2017.
[31] A. Durand, "Deploying IPv6," *IEEE Internet Computing, 5(1)*, 2001.
[32] H. Chan, D. Dash, A. Perrig, and H. Zhang, "Modeling Adoptability of Secure BGP Protocol." SIGCOMM, 2006.
[33] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy," RFC 4632, 2006.
[34] V. Kotronis, X. Dimitropoulos, R. Klöti, B. Ager, P. Georgopoulos, and S. Schmid, "Control Exchange Points: Providing QoS-enabled End-to-End Services via SDN-based Inter-domain Routing Orchestration." ONS, 2014.
[35] D. Farinacci, D. Lewis, D. Meyer, and V. Fuller, "The Locator/ID Separation Protocol (LISP)," RFC 6830, 2013.
[36] J. R. Correa, A. S. Schulz, and N. E. Stier-Moses, "Selfish Routing in Capacitated Networks," *Mathematics of Operations Research, 29(4)*, 2004.
[37] S.-H. Yook, H. Jeong, and A.-L. Barabasi, "Modeling the Internet's Large-Scale Topology," *Proceedings of the National Academy of Sciences, 99(21)*, 2002.
[38] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, "On the Treeness of Internet Latency and Bandwidth." SIGMETRICS, 2009.
[39] F. Kelly, *Mathematical Modelling of the Internet*. Springer-Verlag, 2001.
[40] C. Papadimitriou, "Algorithms, Games, and the Internet." STOC, 2001.

List of Authors

Annex to the IFIP Networking 2018 Proceedings

Demonstration and Poster Session

Annex to ISBN 978-3-903176-08-9 © 2018 IFIP

Demo: Chaining of Segment Routing aware and unaware Service Functions

Ahmed Abdelsalam
Gran Sasso Science Institute

***Abstract*—Segment Routing (SR) is a source routing paradigm that can benefit from both MPLS and IPv6 data planes to steer traffic through a set of nodes. It provides a simple and scalable way to support Service Function Chaining (SFC). In this demo, we propose an NFV architecture based on SR and implemented in Linux environment. It allows chaining of both SR-aware and SR-unaware Service Functions (SFs). In order to include SR-unaware SFs into SR SFC, we use our SR proxy implementation: srext, a Linux kernel module that handles the processing of SR information in behalf of the SR-unaware SFs. As SR-aware SFs, we use two of our implementation; SERA and SR-aware snort. SERA is a SEgment Routing Aware Firewall, which extends the Linux iptables firewall, and capable of applying the iptables rules to the inner packet of SR encapsulated traffic. SR-aware snort is an extended version of snort that can apply snort rules directly to inner packet of SR encapsulated traffic. We show the interoperability between SR-aware and SR-unaware SFs by including both of them within the same SFC.**

***Index Terms*—Service Function Chaining, Network Function Virtualization, Segment Routing, Linux networking**

I. Introduction

Telecommunication networks infrastructures are evolving at a rate rarely seen since the transformation from analog to digital [1]. Network functions virtualization (NFV) offers an agile way to design and deploy networking service [2]. In an NFV infrastructure, network functions are decoupled from proprietary hardware appliances and moved to virtual servers so they can run in software modules called Virtual Network functions (VNFs), which are sometimes referred to as Service Functions (SFs). This dramatically reduces both capital expenditures (CAPEX) and operating expenses (OPEX) [3]. A set of these VNFs (SFs), which can be arbitrarily located in a distributed virtualization infrastructure, are often required to deliver an end-to-end service, hence Service Function Chaining (SFC) comes into play.

SFC denotes the process of forwarding packets through the sequence of SFs [4]. It requires a steering mechanism to force packets to go through SFs. These steering mechanisms often require inserting a new header into packets, which carries the path information. Network Service Header (NSH) and IPv6 Segment Routing header (SRH) are two examples of those headers. NSH has been proposed by the IETF SFC Working Group to support the encapsulation of packets with a header that specifies the sequence of SFs to be crossed [5]. Using NSH requires creating a state (per each NFV chain) in the network fabric, which doesn't make it the preferred solution in the recent era of networking, where everything is going towards stateless and simplicity. On the contrary, using SRH for SFC doesn't have the need for those state information.

In this demo, we consider the use of the Segment Routing (SR) architecture to support SFC. SR is a new network architecture that leverages the source routing paradigm [6]. It allows to steer packets through an ordered list of nodes, which are refereed to as segments. SR can be instantiated over both MPLS (SR-MPLS) and IPv6 (SRv6) data planes. SRv6 defines a new IPv6 Routing type, named SRH. It allows including a list of segments in the IPv6 packet header [7]. Each A segment is encoded as an IPv6 address and represents function to be called at a specific location in the network. SR enables SFC in a simple and scalable manner, by associating each SF with a segment. Such segments are combined together in a segment list to achieve SFC.

II. SR-aware vs SR-unaware SFs

SFs can be categorized into two types, depending on their ability to properly process SR encapsulated packets. These are respectively named SR-aware and SR-unaware SFs [8]. An SR-aware SF is able to correctly process SR-encapsulated packets it receives, which imply being able to process the original packet despite the fact that it has been encapsulated within a SR packet, but also being able to process the SRH. On the contrary, An SR-unaware SF is not able to correctly process the SR-encapsulated it receives. It may either drop the traffic or take erroneous decisions due to the unrecognized SR information. In order to include SR-unware SFs in an SR SC policy, it is thus required to remove the SR information as well as any other encapsulation header before the SF receives the packet, or to alter it in such a way that the SF can correctly process the packet. SR proxy is an entity, separate from the service, that performs these modifications and handle the SR processing on behalf of a SR-unaware service. Srext [9] is a Linux kernel module providing advanced SR functions. It supports different SR proxy behaviours detailed in [8].

III. SR/SFC Testbed

In order to showcase the SFC of SR-aware and SR-unaware SFs, we built the testbed shown in Figure 1, which consists of six nodes (R1-R6), implemented as Linux VMs and represent our SR domain. This SR domain is used to connect two branches (BR1 and BR2) of an enterprise to an external network (Ext). All nodes, except R4, support SRv6. Nodes R1 and R6 respectively represent the ingress and egress nodes, while nodes R2, R3 and R5 are used as NFV nodes of our

SRv6 based SFC scenario. Node R4 is a normal IPv6 Linux router. Service Functions (F1-F3), Branches (BR1 and BR2), and external network (Ext) are deployed as Linux network namespaces. F1 is an SR-aware Linux iptables firewall, F2 is SR-unaware snort, and F3 is an extended SR-aware version of snort. Nodes R1, and R4-R6 are running kernel 4.14 and have iproute2 v4.14 installed. Node R2 is running compiled Linux kernel 4.15-rc2 with SRv6 enabled and SERA firewall included [10]. Node R3 has the srext [9] kernel module installed. The links between any two nodes Rx and Ry are assigned IPv6 addresses in the form `fc00:xy::x/64` and `fc00:xy::y/64`. For example, the two interfaces of the link between R1 and R2 are assigned the addresses `fc00:12::1/64` and `fc00:12::2/64`. Each node owns an IPv6 prefix to be used for SRv6 local SID allocation, which is in the form `fc00:n::/64`, where `n` represents the node number. As an example, R2 owns the IPv6 prefix `fc00:2::/64`. SFs are instantiated on an SR SID of form `fc00:n::fk:/112` where `n` represents the node hosting the SF and `k` is the SF number. For example, F1 which is running in node R2 is given the prefix `fc00:2::f1:/112`. BR1, BR2, and Ext are respectively assigned the IPv6 prefixes `fc00:b1::/64`, `fc00:b2::/64`, and `fc00:e::/64`.

IV. SR/SFC POLICIES

The testbed in Figure 1 supports two different path, with different bandwidth and security guarantees, towards *Ext*. Path p_1 ($R_1 \rightarrow R_4 \rightarrow R_5 \rightarrow R_6$) provides high bandwidth. Path p_2 ($R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_6$) has lower bandwidth, but more security guarantees. Going through p_1 implies crossing F1 and F2. The same way, going through p_2 implies crossing F3. BR1 and BR2 have different traffic requirements; BR2 traffic is very delay-sensitive, while BR1 traffic is highly confidential, but less delay sensitive. We exploit p_1 and p_2 to satisfy those traffic requirement. BR1 traffic is steered through p_1, and BR2 traffic is steered through p_2. At the ingress node (R1), we configured two different SR SFC policies (CP1 and CP2) that steer traffic through p_1 and p_1. Policy Based Routing (PBR) is used to classify traffic coming form BR1 and BR2, which respectively go through CP1 and CP2.

V. DEPLOYMENT AND TESTING

We built our testbed by using *VirtualBox* [11] as hypervisor and *Vagrant* [12] as VM manager. This makes it easy to replicate the demo on any commodity hardware. Scripts required to deploy the demo are open source and can be found at [13]. To verify the deployment of the demo, we use *iperf* [14] to generate traffic from BR1 and BR2. BR1 traffic should cross both F1 and F2. F1 is configured with iptables rules, which are applied by SERA firewall directly to inner packet of received SR traffic. F2 is an SR-unaware snort, which can't correctly processes SR packets. We used srext to remove SR encapsulation from packets before being handed to F2. The removed SR encapsulation is re-added again to packets after being processed. F3 is the only SF crossed by BR2 traffic. It's an SR-aware snort that can apply configured snort rules directly to inner packet. To make sure that BR1 and BR2 traffic follows the exact path in both upstream and downstream, we configure two SR SFC policies on the egress node (R6) for the reverse path. This guarantees that SFs get the traffic in both directions.

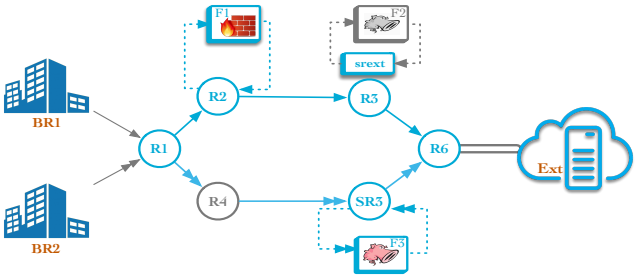


Fig. 1: Testbed for SR/SFC demo

VI. CONCLUSIONS

In this demo, we introduce a Linux NFV infrastructure that support SFC of both SR-aware and SR-unaware SFs. We used our SR proxy implementation (srext) to include those SR-unaware SFs in an SR SFC. As SR-aware SFs, we provided two implementations of SR-aware SFs. SR-aware and SR-unaware SFs have been included in the same SFC to show their inter-operability. We provided an open source implementation for the SR/SFC testbed been used.

REFERENCES

[1] B. Thekkedath, *Network Functions Virtualization For Dummies*. Wiley, 2016.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[3] R. Mijumbi et al., "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, 2015.

[4] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[5] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," Internet-Draft, November 2017. [Online]. Available: http://tools.ietf.org/html/draft-ietf-sfc-nsh

[6] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The Segment Routing Architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[7] S. Previdi (ed.) et al., "IPv6 Segment Routing Header (SRH)," Internet-Draft, September 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-02

[8] F. Clad *et al.*, "Segment Routing for Service Chaining," Internet-Draft, October 2017. [Online]. Available: https://tools.ietf.org/html/draft-clad-spring-segment-routing-service-chaining-00

[9] "srext - a Linux kernel module implementing SRv6 Network Programming model," Web site. [Online]. Available: https://github.com/netgroup/SRv6-net-prog/

[10] "SERA - SEgment Routing Aware Firewall," Web site. [Online]. Available: https://github.com/SRouting/SERA

[11] "VirtualBox home page," Web site. [Online]. Available: http://www.virtualbox.org/

[12] "Vagrant home page," Web site. [Online]. Available: http://www.vagrantup.com/

[13] "SRv6 SFC demo," Web site. [Online]. Available: https://github.com/SRouting/sr-sfc-demo

[14] "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," Web site. [Online]. Available: http://iperf.fr

Demo: Intent-Based 5G IoT Application Slice Energy Monitoring

*+Fred AKLAMANU, *Sabine RANDRIAMASY, +Eric RENAULT, *Imran LATIF, *Abdelkrim HEBBAR, *Alberto CONTE, *Bilal AL_JAMMAL, *Warda HAMDAOUI
*Nokia Bell Labs, Nozay, France
firstname.lastname@nokia-bell-labs.com

+Institut Mines-Télécom / Télécom SudParis, Samovar CNRS, France
firstname.lastname@telecom-sudparis.eu

***Abstract*—Current Telco Network Service Provisioning requires proficient expertise on Infrastructure equipment. Moreover the process is tedious and erroneous making Network Service lifecycle a daunting task for Network Operators (NOs) and 3rd Party Network Tenants. This paper, proposes an Over-The-Top Intent Based Network Framework for Network Slice Energy Monitoring and Provisioning. The aim is to automate the task of network slicing through a declarative approach known as Intents while hiding network complexity enabling NOs monitor Network Slice energy consumption. We provide an experimental validation of the Intent Framework with a scenario of a 5G IoT Application Network Slice energy monitoring.**

I. Introduction

The next generation mobile network, 5G is becoming a reality. The backbone or foundation of 5G is viewed as Software Defined Networking [1] and Network Function (SDN) Virtualisation (NFV) [2]. These technologies provide potential cost cutting .i.e. Capital Expenditure (CAPEX), Operational Expenditure (OPEX) and they will help Network Operators (NOs) maintain elastic networks to meet growing network demands.

5G will potentially provide a window for NOs to extend their infrastructure services to Mobile Virtual Network Tenants (MVNO) and Vertical Markets such as Over-The-Top Application Providers. The services envisioned by NOs to potential tenants will include physical or virtual shared end-to-end network resources known as a Network Slices. This encompasses network resources from Cloud Radio Access Network (CRAN) [3] through to Evolve Packet Core Network (EPC) [4] which traverses a transport network. Both NOs and Network Tenants will want an idea of how much energy these network slices consume. This will help NOs to bill their clients based on network slice energy consumption and other factors.

The current approach to realise network services by NOs requires proficient expertise on infrasturucture equipments and moreover the process is tedious and erroneous. They exert a manual method to deliver the network services to their clients i.e. MVNOs and vertical markets. Such an approach will not suffice on 5G mobile networks due to a 30 million expected connected devices. Intent-Based Networking (IBN) [5] aims to ease such challenge of manual network service provisioning through network automation. Networks Tenants will only have to specify their Intents i.e. "WHAT" network service while an Intent-Based Management Framework handles the automation process of realising the Intent, that is, the "HOW". An Intent in this paper refers to an Over-The-Top (OTT) Network Application Slice.

We propose an OTT Intent-Based Network Framework which provides a simplified and non-technical interface for MVNOs and vertical markets to request for OTT Network Application Slice without knowledge of the physical insfrastructure details such as configuration, topologies and protocols. The framework enables energy monitoring of such Network Application Slices.

II. Proposed Over-The-Top Intent Based Networking (IBN) Framework

The proposed IBN Framework provides network tenants and NOs with a simple service platform. The framework speeds up service request placement and provisioning, provides a feedback for network service feasibility and guarantees the platform reliability. The modules of the OTT IBN platform on top of a Cloud-Over-The-Top Application Slicing Platform (COASP) are shown in Fig. 1.

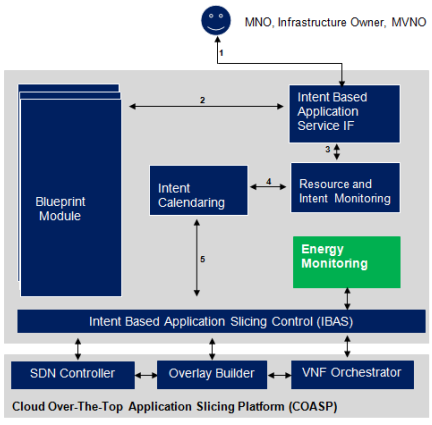


Fig. 1. OTT IBN Framework

We provide details on the energy monitoring module for VNFs on the OTT IBN Framework. Details of other

sub-modules of the framework are skipped here and provided in our related work on Intent-Based Real Time 5G Cloud Service Provisioning. We employ the term micro-services for VNFs. These micro-services are cloud native Docker containers [6] which provide numerous advantages over traditional virtual machine. They are lighter and setup time is in order of few seconds compared to minutes for virtual machines.

A. Energy Monitoring Module

This module is responsible for microservices energy consumption monitoring. The module comprises a Software Defined Network middleware power metering, PowerAPI [7]. This tool enables the power measurements of micro-services which will be important for NOs about possible VNF placement decisions and furthermore identify high energy consuming ones. The energy data will be potentially useful in the future for data analysis and learning purposes.

III. DEMONSTRATION

The demonstration involves the setup of the physical insfrastructure. This is made up of two Nokia Airframes Front-End Unit (FEU) and Edge Cloud (EC) and a laptop for Central Cloud (CC) interconnected by two switches. The table I shows the physical platform specifications.

	Front End Unit	Edge Cloud	Central Cloud
OS	Ubuntu 16.04	Ubuntu 16.04	Ubuntu 16.04
RAM (GB)	128	128	16
CPU Cores	24	24	8

TABLE I
CLOUD PLATFORMS

The next phase is the virtual network setup, deployment of a VNF Orchestrator, distributed key-value store, ETCD [8] and ONOS [9], an SDN controller as a docker container on the Central Cloud (CC). This phase also involves the installation of Open Virtual Switches (OVS) [10] on FEU and EC. ONOS is responsible for the management of the OVS to establish connectivity for the Virtual Network Infrastructure (VNI).

Phase two involves deployment of 5G helper VNFs, these are VNFs necessary for the deployment of 5G Network Application Slices. 3 VNFs and 5 VNFs are deployed on FEU and EC respectively by a VNF Orchestrator.

The above steps ensure that the VNI is properly set up to receive network service requests. The Network Tenant provides the type of service (Intent), which is a 5G IoT Application Slice from a GUI interface. Energy monitoring option is activated for End-to-End Network Slice monitoring. End-to-End Network Slice monitoring comprise energy monitoring of all micro-services on the VNI. The tenant request is transmitted to the Intent Engine for feasibility of service deployment. In the absence of any problem, a 5G IoT VNF is deployed as well as PowerAPI docker containers for individual microservices monitoring on the VNI. The individual consumption of the VNFs are summed and stored in a real-time database, influxDB. The total network slice energy consumption is displayed on a dashboard as shown in Fig. 2.

Fig. 2. 5G IoT Application Slice Energy Consumption

IV. CONCLUSION AND FUTURE WORKS

Our proposed OTT Intent Based Networking Framework simplifies network slice energy monitoring through automation. The NO does not need to manually setup different configuration for OTT Application Network Slice components for energy monitoring.

V. ACKNOWLEDGMENT

The power metering virtualization was done within the studies of the Celtic-Plus project SooGREEN [11].

REFERENCES

[1] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *ICT Convergence (ICTC), 2012 International Conference on*. IEEE, 2012, pp. 360–361.

[2] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," vol. 51, no. 11. IEEE, 2013, pp. 24–31.

[3] e. Imran Latif, "Cloud ran architecture for smart cities," in *The 1st American University in The Emirates International Research Conference (AUEIRC)*. Springer, 2017.

[4] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," vol. 28, no. 6. IEEE, 2014, pp. 18–26.

[5] SDxCentral, "Intent: Dont tell me what to do! (tell me what you want," February 2015. [Online]. Available: https://www.sdxcentral.com/articles/contributed/network-intent-summit-perspective-david-lenrow/2015/02/

[6] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," vol. 2014, no. 239. Belltown Media, 2014, p. 2.

[7] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the process-level," vol. 2013, no. 92, 2013.

[8] Core OS, "A distributed, reliable key-value store for the most critical data of a distributed system." [Online]. Available: https://coreos.com/etcd/

[9] ON.Lab, "ONOS intent framework," May 2016. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Intent+Framework

[10] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch." in *NSDI*, 2015, pp. 117–130.

[11] SooGREEN, "Service-oriented optimization of green mobile networks" (soogreen), celtic-plus project, partially funded by the french directorate general for enterprise (dge) and the scientific and technological research council of turkey (tubitak)," 2018.

Demo: Enabling In-Network Processing utilizing Nearby Device-to-Device Communication

The An Binh Nguyen, Christian Klos
Björn Richerzhagen, Ralf Steinmetz
KOM, TU Darmstadt, Germany
{firstname.lastname}@kom.tu-darmstadt.de

Christian Meurisch
TK, TU Darmstadt, Germany
meurisch@tk.tu-darmstadt.de

Patrick Lampe
University Marburg, Germany
lampep@mathematik.uni-marburg.de

***Abstract*—In disaster situations, relief work can be enhanced and facilitated by acquiring and processing distributed information. However, the communication and computation infrastructures might be impaired or inaccessible in emergency response scenarios. Consequently, approaches for coordination, and resource utilization are still challenging. To this end, we proposed the concept of an *adaptive task-oriented message template* (ATMT), that bundles the control information and the payload data, required to process and extract information, into a single message. Thus, an ATMT enables distributed in-network processing of complex tasks, allows to leverage the idle resources of mobile devices of the first responders. In this paper, we demonstrate the use of the ATMT concept in an example of face detection, which can be used to offer *Person Finder* similar services in an emergency ad hoc network. We utilize Google Nearby peer-to-peer networking API, standard and available on Android-based devices, to realize the handover of an ATMT message between mobile devices. This successful integration underpins the prospective adoption of the ATMT concept.**

I. Introduction

Acquiring and processing distributed information are crucial for disaster situations. Today, several services designed to enhance relief work, and to offer information relevant for emergency situations, such as Google's Person Finder [1] or Facebook's Crisis Reponse [2], are available. However, these services require stable Internet-based communication, which might not be possible in disaster situations. To maintain communication in emergency situations, mobile hand-held devices such as smart phones can be used to create an opportunistic ad hoc network [1], which allows mobile devices to share data and exchange information through device-to-device communication. Combining with the built-in sensors, and the computing resource available on mobile devices, it is possible to provide emergency relief services on top of mobile opportunistic ad hoc network. Hereby, approaches to enable distributed coordination, and efficient resource utilization are necessary. For this purpose, we proposed and designed a message template, called *adaptive task-oriented message template* (ATMT) in [2].

An ATMT message describes a complex task, the operations and the payload data required to complete the defined complex task; which makes each ATMT message a self-encapsulated message. Hence, the ATMT message template is able to support distributed processing, leveraging idle resource of the participating mobile devices. Additionally, since an ATMT message is self-encapsulated, each participating device can make an autonomous decision based on its available capability, and its available resources. Overall, the ATMT message provides a basis to create complex services on an opportunistic network, utilizing mobile devices.

To showcase the advantages, and the applicability of the ATMT concept in practice, we provide a demonstration, that (i) uses ATMT message template to implement a face detection technique, which requires multiple-processing stages, specially designed for mobile devices [3], and (ii) utilizes the Google Nearby Networking API [4] for enabling handover of an ATMT message directly between devices.

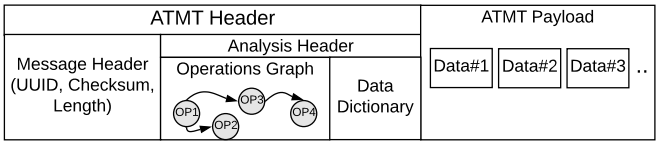


Fig. 1: Structure of the adaptive task-oriented message template (ATMT) as proposed in [2]

II. Adaptive Task-oriented Message Template

The design and construction of the ATMT message template are illustrated in Fig. 1. The goal in designing ATMT message template is to allow a user to define a processing goal, and the steps/operations required to reach this goal. These information are captured in an *operations graph*, modeled by a *directed acyclic graph* within the *ATMT header*. A device can check on the processing status of the *operations graph*, by reading only the *checksum* field, without reading the whole content of the ATMT message. This allows a device to make a fast decision on whether to merge, to drop, or to handover an ATMT task. To incorporate the payload data required for the operations, considering the resource constraints of the devices in an opportunistic mobile ad hoc network, we devise the *ATMT payload* to contain pieces of data, which can be compressed by different encoding methods to allow for more flexibility. Each piece of data is mapped to an operation

[1] https://google.org/personfinder
[2] https://www.facebook.com/about/crisisresponse/

in the *operations graph* through the *data dictionary* in the ATMT header. To organize the coordination among mobile devices, we conceive four roles, which can be assumed by the participating devices according to their available capabilities. These roles are *sensor node* to obtain data, *delegator node* with domain knowledge to set up and adapt the *operations graph* if necessary, *operator node* which executes one or more operations from the operations graph based on its available resource, and *forwarder node* which receives, store and forward an ATMT message. Distributed processing of a complex task is realized by simply passing ATMT messages. Each device, receiving ATMT messages, will act accordingly to its role, adjust the content of ATMT messages w.r.t. the current processing state, and handover the adjusted ATMT message to the other.

III. Integration of ATMT with Google Nearby

Support to setup and to provide device-to-device communication using mobile devices such as smart phones is still restricted. To enable WiFi ad hoc communication between Android devices, these are required to be rooted. Even though WiFi direct allows for direct communication, but it requires a master-slave model, which makes transmission of a message through multi-hops difficult [1]. Recently, Google enables and provides *Nearby* [4], a peer-to-peer networking API that allows for discovery, connection and data exchange with devices in the close vicinity. We use *Nearby Connections* to let each device advertise its role/service. Thus, each participating device is able to look for the next role/service, that it requires. For instance, a *sensor node*, after acquiring data, needs to look for a *delegator node*, so that the *delegator node* can setup and include the corresponding *operation graphs* into the ATMT message. Similarly, a *delegator node* searches for *operator nodes* to execute the operations on the obtained data. If an *operator node* cannot complete the task described in ATMT message alone, this *operator node* will look for further *operator nodes* which possess the capabilities, e.g., special hardware, special algorithms/libraries, to take over the upcoming operations. When an *operator node* notices the completion of the ATMT task, it can look for *forwarder nodes* to transport the final result to a predefined destination. In this manner, a multi-stage processing is realized through multi-hops device-to-device communication.

IV. Scenario and Demonstration

We use the ATMT concept to implement the face detection technique proposed in [3], which can be used to support *person finder* service in an emergency ad hoc network. The face detection technique as introduced in [3] relies on multiple stages pipeline to detect multiple faces within an image; these are (1) preprocessing to handle parameters, (2) a fast face detection to determine important regions within the image, and (3) a validation phase using *dlib* library to detect and validate the faces within the image.

Despite the fact, that the face detection technique in [3] is designed considering the resource and energy constraint of a mobile device, this technique can be further enhanced using the ATMT concept. The processing pipeline as described can be distributed to multiple devices; consequently, each device has to process only one phase of the pipeline. The advantages for the utilization of the ATMT concept in this scenario are (i) further reducing the resources consumption of the participating devices, since now each device has to process only one phase of the pipeline, and (ii) being able to leverage heterogeneous capabilities, for instance, in the described scenario, the *dlib* library might not available on all devices.

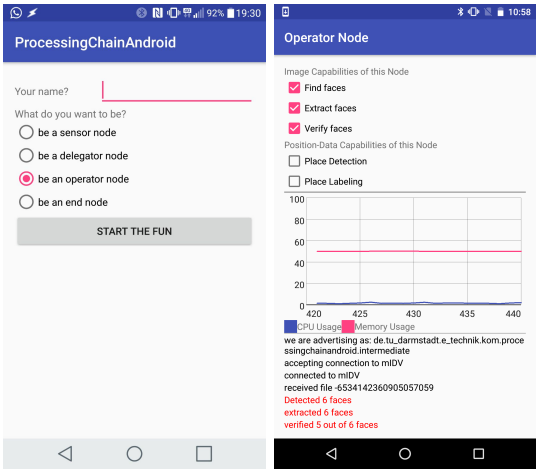


Fig. 2: User interface of the Android mobile devices used in the demonstration

In the demonstration, a user can use a *sensor* device to capture and send a picture containing multiple faces to other devices for further processing. Next, the device chosen as a *delegator node* should receive the picture, adds the *operations graph* for the face detection technique accordingly, and forwards the constructed ATMT message to the *operator devices*. On each *operator* device, the user can choose which operations should be available, and can observe through log messages, how the devices handle ATMT messages, and how the *Nearby Connections* are used (cf. Fig. 2). At the end of the processing pipeline, the results of the face detection can be seen in the chosen end-node device, showing the cropped images of different detected faces. A short video, showing the demonstration as described, can be found at the following link http://bit.ly/2GGenHZ

References

[1] S. Trifunovic, M. Kurant, K. A. Hummel, and F. Legendre, "Wlan-opp: Ad-hoc-less opportunistic networking on smartphones," *Ad Hoc Networks*, vol. 25, 2015.

[2] T. A. B. Nguyen, C. Meurisch, S. Niemczyk, D. Böhnstedt, K. Geihs, M. Mühlhäuser, and R. Steinmetz, "Adaptive Task-oriented Message Template for In-Network Processing," in *NetSys'17*. IEEE, 2017.

[3] P. Lampe, L. Baumgärtner, R. Steinmetz, and B. Freisleben, "Smartface: Efficient face detection on smartphones for wireless on-demand emergency networks," in *IEEE ICT*, 2017.

[4] "Google Nearby." [Online]. Available: https://developers.google.com/nearby/

DEMO: Controlling software router resource sharing by fair packet dropping

Vamsi Addanki, Leonardo Linguaglossa, Jim Roberts, Dario Rossi
Telecom ParisTech, Paris

***Abstract*—We demonstrate a practical way to achieve multi-resource sharing in a software router, where both bandwidth and CPU resources may be bottlenecks. Our main idea (published in a same-titled paper in this year IFIP Networking conference [1]), is to realize per-flow max-min fair sharing of these resources by wisely taking drop decisions according to the state of a shadow system. We implement our FairDrop proposed algorithm in Vector Packet Processor (VPP), a novel high-speed software router architecture. We demonstrate FairDrop is capable of fairly sharing CPU cycles among flows with heterogeneous computing workload, at 10Gbps on a single core.**

I. Introduction

Controlling how bandwidth is shared between concurrent flows is a classical issue in networking, and the advantages of imposing fairness have been repeatedly discussed since Nagle's pioneering work [2]. More recently, the blending of networking and computing raise new challenges [3] in terms of resource contention and sharing – however, simple mechanisms that are capable of handling heterogeneous resources have yet to appear. In emerging high-speed software routers, flow throughput may additionally be impeded by network capacity limitations as well as other resources, such as the amount of available CPU cycles to process packets of any given flow: in this case, it would be desirable in this case to impose per-flow fair throughput expressed in cycle/s[4].

As in[4], we advocate that flexible dropping algorithms are an attractive solution to control resource sharing, be it cycles of a multi-core CPU or network bandwidth. We implement a simple and practical algorithm, which we refer to as FairDrop (FD), that realizes max-min fair flow rates while retaining the network interface card (NIC) and server code optimizations that are necessary to keep up with line speeds of 10 Gbps on a single CPU core. These optimizations notably require packets to be batched for both I/O and processing making implementation of classical scheduling algorithms like DRR [5] problematic if not impossible, as argued in [3].

Our proposal is then to realize fairness via a *shadow system.* Briefly, suppose packets are handled simultaneous by two service systems, one the actual buffer management system implemented in the router (e.g., a DPDK circular ring), the other a shadow system implementing a more sophisticated scheduler (e.g., per-flow FQ). Packets that are dropped in one system are also dropped by the other so that both systems yield exactly the same rate over the lifetime of a flow. The shadow system in our proposal is virtual and makes dropping decisions based on a measure of per-flow virtual queue occupancy. This measure is depleted between packet arrivals, at a rate that varies depending on the number of active flows, and incremented by packet length on the arrival of every batch. In particular, if the shadow system implements per-flow head-of-line processor sharing, the long-term flow rates will be max-min fair.

We implement the above proposal in Vector Packet Processor (VPP), an software router released as open source in the context of the FD.io Linux foundation project. For a detailed explanation of our FairDrop (FD) algorithm we refer the interested reader to a same-titled paper in this year IFIP Networking conference [1]. In this extended abstract we instead describe the experimental environment and scenarios that we will demonstrate, contrasting results achieved under simple buffer management policies (such as FIFO or NIC ring buffers). More information about the project, as well as our implementation, is available at [6].

II. FairDrop implementation and demonstration

In a software router, a CPU core becomes a bottleneck when flows emit packets too fast yielding a compute load greater than the CPU capacity, leading to packet drops. High-speed software routers are intrinsically flow-aware: flow-awareness is facilitated by NICs implementing receive side scaling (RSS), that hashes the 5-tuple and maps packets to distinct virtual queues, mainly for the purpose of load balancing over multiple CPU cores. Individual threads of packet processing applications are bound to a CPU core and, using kernel-bypass stacks such as DPDK, threads consume independent streams of packets, each from a different RSS queue. Additionally, high-speed software routers and their NICs generally deal with packets in *batches* rather than individually, which reduces interrupt pressure and that is a necessary optimization for line-speed packet processing. Software routers typically polls for available packets in the NIC circular buffer, grabbing and processing the whole batch before the next poll. FairDrop operates over packet batches at the router ingress.

We demonstrate FairDrop with a scenario where N flows share a C=10Gbps link and are processed by a single CPU core clocked at 2.6GHz. Particularly, flows have equal input rate C/N but different treatment cost. For the sake of simplicity, in the demonstration we consider only two flow classes: the majority of the flows belong to the light-weight class C_L (e.g., Ethernet switching or IPv4 forwarding), whereas few flows belong to a heavy-weight treatment class C_H (e.g., IPsec or stateful L4 operation). In particular, we select functions whose $C_H/C_L \approx$10 so that a single packet of an heavy-weight flow

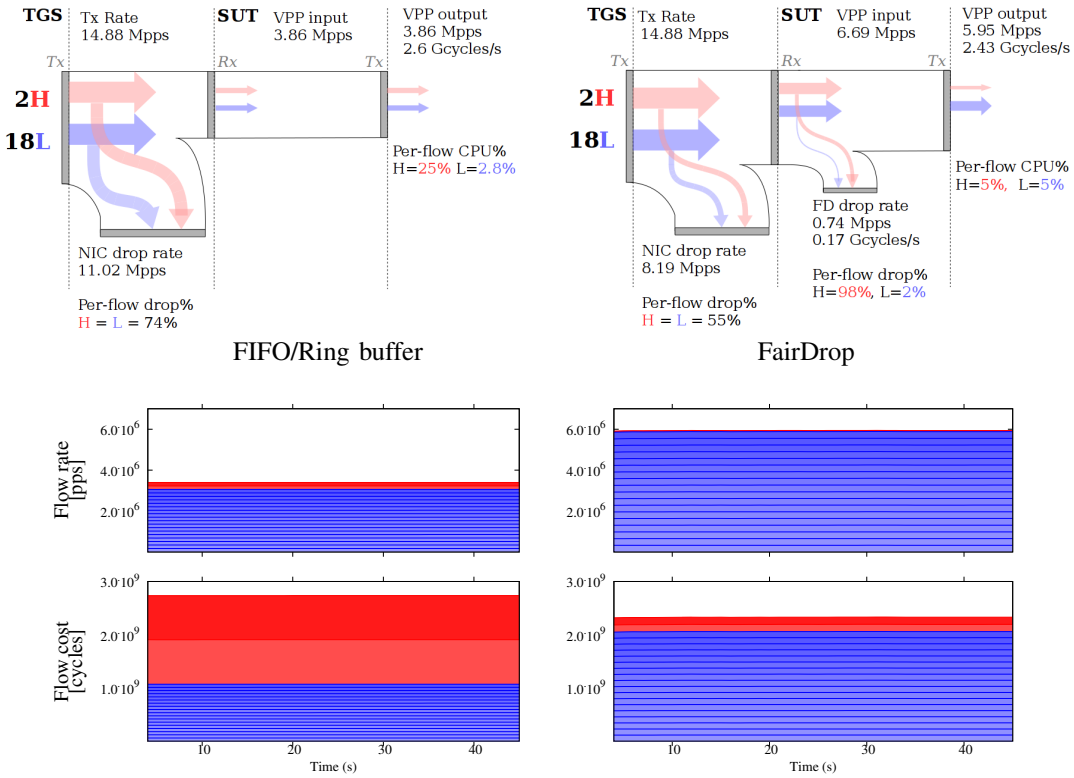


Fig. 1. Illustration of Classical (left column) vs FairDrop (right column) operations. Top part reports sankey diagrams of the rate at the Traffic Generator (TGS) and at the System Under Test (SUT). Lower part depicts the time evolution of the flow rate (in Mpps, middle) and the flow ccost (in cycles, bottom).

requires as many CPU cycles as about 10 packets of light-weight flows. We additionally fix $N_H = 2$ and $N_L = 18$ so that out of the total $N = 20$ flows, the N_H flows of class C_H requires as many processing cycles as the N_L flows of class C_L. Needless to say, 64B packets are sent to the maximum rate of 14.88Mpps, so that not all flows can be processed with the CPU budget.

We represent experimental results of the demo with the visual layout of Fig.1, where plots in the left column represent the case of traditional buffer management, and plots in the right column report the FairDrop case. In particular, the top plots report a sankey visualization of the experiments, whereas the bottom plots report the individual flow rate (in packets per second) and the individual flow cost (in cycles per second). The two heavy-weight flows are represented in red, and the 18 light-weight flows in blue.

In the traditional case, since the CPU budget is not enough to process packet of all flows, about 74% of packets are lost at the NIC before entering the VPP router. Given that flows have equal rates, there is no loss differentiation at the NIC, so that only about 3.86Mpps exit the VPP router, consuming the 2.6Gcycles/sec budget of our CPU. Notice that each flow have equal rate, but that a single heavy-weight flow alone consumes 25% of the CPU budget.

Conversely, the FairDrop mechanism preferentially drops packets of the heavy-weight flows to reinstate fairness (at a rate approximately 10 times higher). Dropping decisions have a cost (i.e., the packets need to be fetched from the NIC, the queue in the shadow system is updated, etc.) and FairDrop consumes 0.17Gcycles/sec. The net result of fair dropping decisions, more light-weight packets are processed in the router: this increases the overall throughput at 5.95Mpps (top right plot), reducing the drops at the NIC buffer, and reinstates per-flow fairness in terms of the number of cycles (bottom right plot).

The demonstration will allow to interact with the VPP router configuration (e.g., FairDrop vs classical ring management) and altering the scenario parameters (e.g., number of flows, relative cost, etc.) to contrast the key performance indicators under both approaches.

Acknowledgments

This work was funded by NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech.

References

[1] V. Addanki, L. Linguaglossa, J. Roberts, and D. Rossi, "Controlling software router resource sharing by fair packet dropping," in *IFIP Networking*, 2018.
[2] J. Nagle, "On packet switches with infinite storage," RFC 970, 1985.
[3] K. To, D. Firestone, G. Varghese, and J. Padhye, "Measurement based fair queuing for allocating bandwidth to virtual machines," in *ACM HotMiddlebox*, 2016.
[4] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *ACM SIGCOMM Comput. Commun. Rev.*
[5] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *SIGCOMM Comput. Commun. Rev.*
[6] https://newnet.telecom-paristech.fr/index.php/fairdrop/.

Poster: Impact of Prioritized Network Coding on Sensor Data Collection in Smart Factories

Marie Schaeffer, Roman Naumann, Stefan Dietzel, and Björn Scheuermann
Humboldt-Universität zu Berlin, Germany
Email: {marie.schaeffer, roman.naumann, stefan.dietzel}@hu-berlin.de, scheuermann@informatik.hu-berlin.de

***Abstract*—Utilizing information from the production process is integral to smart factory concepts. In our example use-case, plastic industry, sensor information from the injection molding process helps to detect defective parts and provides automated guidance for process set-up. An enabler for such applications is a means to wirelessly collect machines' sensor information in harsh factory environments, and network coding has been proposed as a tool to implement suitable network protocols. Using pre-recorded sensor data from actual injection processes, we study the impact of network coding on the latency of sensor data collection. In particular, we show how network coding with prioritization helps to reduce delays until information becomes usable.**

I. Introduction

Many smart factory use cases strive to automate previously manual tasks via the utilization of highly detailed process information. In our example use-case, plastic injection molding, molten plastic is injected with high pressure and temperature into a form, termed the "mold." As the plastic cools down, the final product hardens out and is finally ejected from the mold. Here, relevant process information includes material pressure and temperature measured within the mold. Such information, in combination with machine learning techniques, allows the automated detection of a variety of product defects before they can reach the customer [1], [2].

In order to leverage process information, it has to be collected quickly from machines throughout the factory. A centralized server then acts upon results and, for example, issues alarms to operators should the process become unstable. Wireless transmission of sensor information is preferable, because it avoids expensive retrofitting of factories. Wireless transmission, however, can be difficult due to the harsh factory environment with metal obstruction and widespread factory areas that necessitate multi-hop capabilities.

Using network coding in our use case can improve the throughput, simplify routing decisions, and add robustness against packet loss. But using random linear network coding (RLNC) to transmit sensor information may result in intolerable delays due to the "all-or-nothing" property. This property states that it is highly unlikely that the server can decode parts of the sensor information before a sufficient number of linear combinations for, in our case, a complete injection cycle are received. A number of prioritized network coding schemes have been proposed to allow early decoding of a subset of a generation's information.

We study the impact of two prioritized network coding techniques – hierarchical network coding (HNC) [3] and iNsPECt [4] – on delays in sensor data collection. As a third mechanism, regular RLNC [5] serves as a baseline for our comparison.

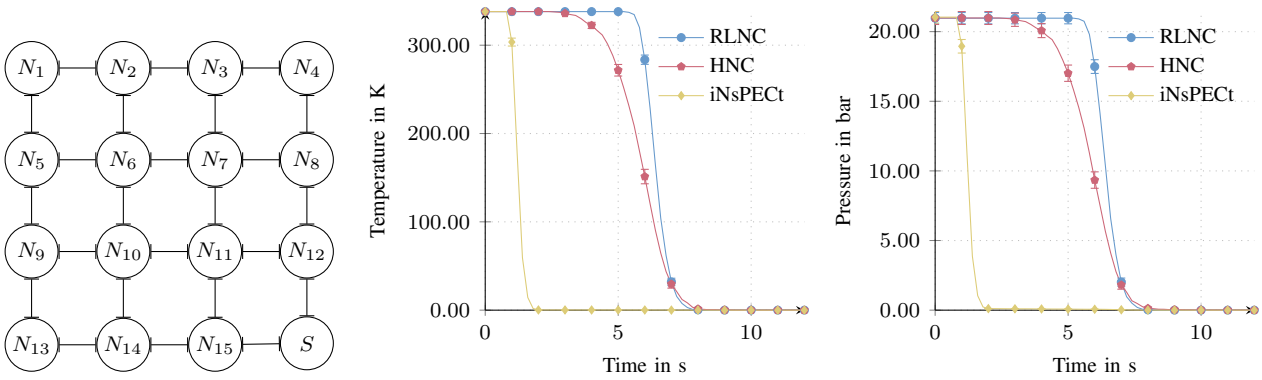
II. Encoding and Transmission Schemes

Using regular RLNC as an example, we explain how network coding in general can be applied to our sensor data collection use case. We then briefly introduce the two prioritized network coding mechanisms used in our comparison.

RLNC splits information into generations of data messages. In our case, a generation is one production cycle's worth of sensor information from a single sensor. Each message is a set of sensor samples and consists of several symbols over a finite field. We employ the common finite field $\mathbb{F}_{2^8}$, as it combines efficient byte alignment with sufficient protection from linear dependency. Each machine generates linear combinations of one generation's messages using *random* coefficients. Each machine then continually broadcasts these linear combinations until all neighbors can decode the current generation. Subsequently, the next generation is sent.

To apply prioritized network coding techniques to our industrial use case, we pre-process sensor information such that it can be divided into different priority layers, as described in [6], We apply discrete cosine transform (DCT) to each production cycle's sensor information and divide its output into blocks of coefficients. Blocks with low-frequency coefficients provide an early preview of a complete sensor cycle, whereas blocks with high-frequency coefficients incrementally increase precision to enable more demanding detection techniques. We again use one injection cycle as a generation, but we use blocks of coefficients as the prioritized network coding mechanisms' prioritization layers. To generate a linear combination associated with a given priority layer, the prioritized network coding (PNC) codes combine only messages of equal-or-lower layers. In our case, this concept translates to only lower-or-equal frequencies of the DCT-provided spectrum of sensor information. As the prioritized layers form a linear subspace in the decoding matrix, they can generally be decoded earlier and, therefore, reduce delays in data processing.

In our evaluation, we study the impact of HNC [3], a PNC protocol, on the delay after which information is usable by the central server. We also study the impact of layer selection, a central aspect of PNC protocols, on decoding delay. To that extent, we compare HNC, which selects priority layers at random, with iNsPECt [4], which employs limited knowledge



(a) Grid topology with one sink.

(b) Temperature error.

(c) Pressure error.

Fig. 1. Topology and average sensor error over time.

on neighboring network nodes' decoding states to determine ideal layers.

III. Factory Network Model and Evaluation

We use a wireless network model in which nodes broadcast their messages. Our topology, given in Figure 1a, represents a typical factory layout with rows of machines in a regular grid and node distances of 30 m. The fifteen nodes, N_1 to N_{15}, represent the machines where the sensor data is measured. One sink node, S, is the factory's central server system. We consider a single sensor for each machine. More sensors in machines bring a constant factor for the amount of required transmissions, analogous to a higher sample rate.

We evaluate using the discrete event network simulator ns-3 (version 3.25) with YANS Wifi model, 802.11g MAC, and 2.4 GHz PHY using log-distance propagation loss model ($\gamma = 3.0$, which is in line with a range modern factory environments [7]) combined with Rayleigh fast fading. We use real, pre-recorded sensor information from the injection molding process. Our sensor information stems from a 25 s long production cycle that was sampled at 500 Hz rate. Each measured sample is a 4 B floating-point number. We split frequency components into five priority layers with a generation size of 53 frequency components to limit each data message's size to 1008 B. For the PNC-iNsPECt variant, we set the data-feedback ratio to $1 : 2$. Each sample shown in the following is the average over five simulation runs of 200 s simulated duration each, using different sub-streams of ns-3's PRNG. Error bars depict 95% confidence intervals (assuming normal distribution), but might not be visible if the error is negligible. During each run, several production cycles are transmitted to the sink.

Figures 1b and 1c show the simulation results for temperature error over time and pressure error over time. The time measurement starts with the first message being transmitted, which explains the initially very high average error that results from production cycles without any frequency components decodable at the server. Generally, it can be seen that the preview provided by the PNC scheme iNsPECt quickly gains precision and is virtually indistinguishable from the original sensor information much earlier than RLNC can provide any information. HNC also gains precision more quickly on average than RLNC. The overhead of the HNC scheme, however, results in RLNC providing the full picture before HNC can lower the remaining error below 1 K or 1 bar. In contrast, PNC achieves such a low average error approximately four times as fast as RLNC for both temperature and pressure readings. The maximum time until each production cycle was available with full precision was 8.40 s with our baseline RLNC. As a result of the principal message overhead imposed by PNC schemes, iNsPECt and HNC required up to 9.20 s and 17.80 s, respectively, until the preview reached full precision. Especially with iNsPECt, however, the error is extremely low during the time after which RLNC finished transmission.

IV. Conclusion

We studied the impact of prioritized network coding for smart factory use-cases using real sensor information from plastic industry. Our results suggest that iNsPECt provides significant benefits over non-prioritized RLNC, whereas HNC can only provide a coarse preview before RLNC provides the full picture.

References

[1] B. Ozcelik and T. Erzurumlu, "Comparison of the warpage optimization in the plastic injection molding using ANOVA, neural network model and genetic algorithm," Feb. 1, 2006.

[2] H. Oktem, T. Erzurumlu, and I. Uzman, "Application of Taguchi optimization technique in determining plastic injection molding process parameters for a thin-shell part," 2007.

[3] K. Nguyen, T. Nguyen, and S. c Cheung, "Peer-to-peer streaming with hierarchical network coding," in *2007 IEEE International Conference on Multimedia and Expo*, Jul. 2007.

[4] M. Schaeffer, R. Naumann, S. Dietzel, *et al.*, "Hierarchical Layer Selection with Low Overhead in Prioritized Network Coding," in *2018 IFIP Networking Conference (IFIP Networking)*, 2018.

[5] T. Ho, R. Koetter, M. Medard, *et al.*, "The benefits of coding over routing in a randomized setting," 2003.

[6] R. Naumann, S. Dietzel, and B. Scheuermann, "INFLATE: Incremental wireless transmission for sensor information in industrial environments," in *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, Dec. 2015.

[7] S. Phaiboon, "Space Diversity Path Loss in a Modern Factory at frequency of 2.4 GHz," *WSEAS Transactions on Communications*, 2014.

Poster: Design and Evaluation of a Time Efficient Vertical Handoff Algorithm between LTE-A and IEEE 802.11ad Wireless Networks

Sina Rafati Niya, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
Email: [rafati | stiller@ifi.uzh.ch]

***Abstract*—LTE-A and IEEE 802.11ad are two of the networks that can have complementary roles in 5G networking. Thus, this paper proposes a time efficient, predictive, and dynamic Handoff (HO) algorithm between these two protocols. The algorithm presented measures the Signal to Interference Plus Noise Ratio (SINR), the Reference Signal Received Quality (RSRQ), velocity, and the Time of Stay (ToS) for pedestrian mobile users and calculates the best Time to Trigger (TTT) value accordingly. One of the main advances of this algorithm is that the TTT value is calculated dynamically regarding user's velocity and Handoff Failure Ratio (HoFR). Comparisons with other algorithms in this area determine that the algorithm proposed gains the least HoFR for these two protocols by avoiding unnecessary handoffs.**

I. Introduction

One of the major research areas in 5G is the offloading process. With offloading, user traffic traverses the local wireless AP instead of utilizing a continuous connection to cellular networks even in indoor areas. In case of using LTE-A in 5G, a proper network for offloading LTE-A communications with up and downlink data rates of higher than 1 Gbps needs to support the same data rates, otherwise, users will not be willing to switch to the local networks. One of the wireless technologies to support high data rates and being deployed in indoor areas as a replacement of traditional WiFi networks is the IEEE 802.11ad standard. This protocol is known as WiGig because of the Gigabit scale data rates it supports. WiGig provides almost 7 Gbps for downlink and almost 3 Gbps for uplink [2].

Switching the user's network from a home (already connected) network, to a new target network (one of the possible networks to be switched to) and vice versa is known as Handoff (HO) or Handover. In this work, a new time-efficient HO algorithm is designed to provide specifically a seamless connection and offloading between LTE-A and WiGig networks as two of the networks might be used broadly in 5G for high data rates . Results of a comparison with other HO algorithms reveal (cf. Section IV) that the algorithm proposed is capable of reducing Handoff Failure Rates (HoFR) in a predictive fashion. Also, the Time to Trigger (TTT) is updated frequently based on measures defined, and the HO process follows a cross-layer algorithm to increase the time efficiency.

II. Simulation Scenarios, Parameters and Evaluation

Simulation of the scenarios done in Matlab. The focus of this work laid on two scenarios: (1) HO process of a user connected to the LTE-A cell and moves toward the WiGig network as presented in Figure 1. (2) HO process of a user connected to the WiGig network and moves toward the LTE-A cell as presented in Figure 2. Parameter used in simulations are listed in Table 1.

TABLE I
Simulation Parameters

Simulation Parameters	Symbol	Value
eNB Number	N_{LTE}	1
WG-AP Number	N_{WG}	1
Simulation Duration	T_{sim}	1000 s
RSRQ Threshold	$RSRQ_{th}$	19.5 dBm
SINR Threshold	$SINR_{th}$	25 dB
LTE-A eNB Transmission Power	P_{LTE}	30 dBm
WG Transmission Power	P_{WG}	10 dBm
LTE-A Bandwidth	B_{LTE}	100 MHz
WG Bandwidth	B_{WG}	2160 MHz
LTE-A Antenna Height	h_{eNB}	40 m
WG Antenna Height	h_{WG}	1.5 m
Mobility Model	———	Gauss-Markov and LPP
Initial TTT	TTT_i	0.1 s
LTE Frequency	f_{c-LTE}	2100 MHz
WG Frequency	f_{c-WG}	60 GHz
LTE Radius	LTE_r	Whole Simulation Area
User Velocity	V_u	$[0-5]m/s$
Data Transfer Direction	———	Downlink
Number of LTE users	N_{U-LTE}	(1-50)
Number of WG users	N_{U-WG}	(1-50)

The HO algorithm proposed is memory-and-time efficient, and managed in a cross-layer fashion. Number of unnecessary HO and HoFR are managed by TTT value. Being a proactive algorithm, users' mobility, including their next location, speed, and angle of movement, are estimated using the Gauss-Markov mobility model, which is updated and readjusted by the accurate data received from Location Positioning Protocol (LPP). This update increases the precision of the Gauss-Markov model for upcoming estimations.

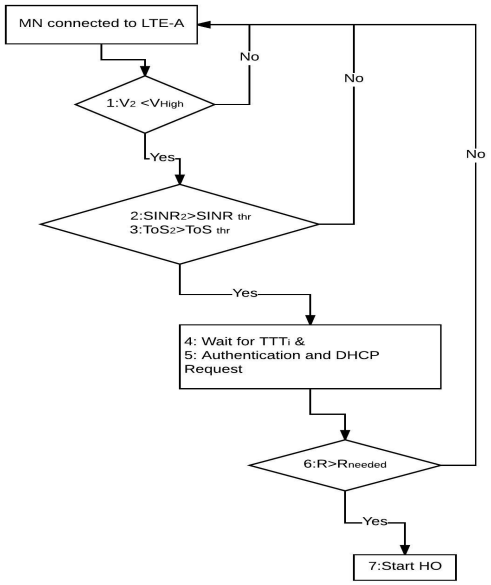


Fig. 1. HO Process: Moving From LTE-A To WiGig

To be computationally efficient, this algorithm does not continuously gather information from target and host networks, instead time intervals are set dynamically according to the amount of TTT to gather information from networks. The algorithm reacts to HoFR increase by adjusting the TTT value. Besides lowering the computational complexity, a binary search is used in TTT calculations to provide faster converges than with a linear method by the order of O(log(n)), which leads a very practical application.

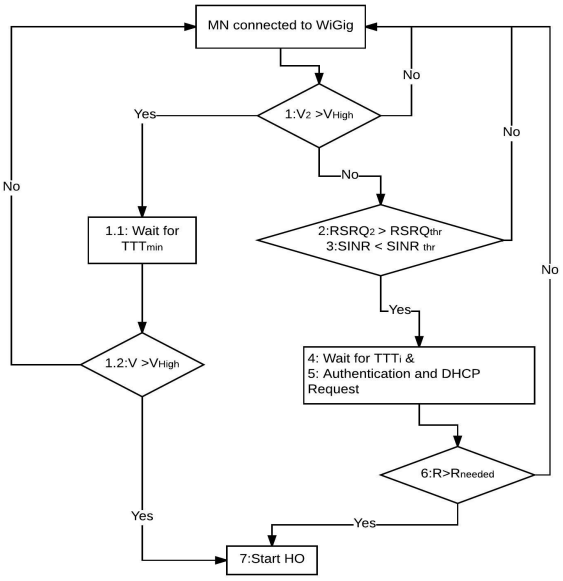


Fig. 2. HO Process: Moving From WiGig to LTE-A

To avoid continuous monitoring of various parameters, this algorithm specifies a high priority to users' velocity. For that reason, checking other variables such as SINR, RSRQ, and TTT is done only, if the user's velocity is in a specific range. Being sensitive to the user's velocity, this new algorithm performs better in comparison to other HO algorithms, especially within the decision making phase for high user velocities.

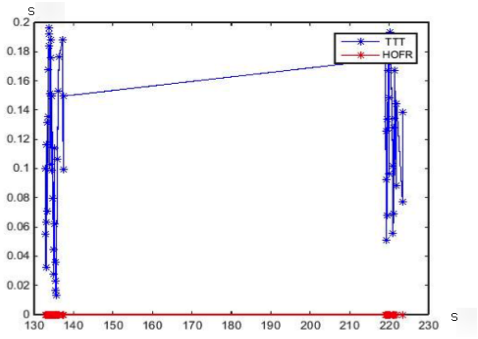


Fig. 3. TTT Variation Using Method of [3] With User Velocity Larger than 3 m/s.

In this algorithm, base (home) and target networks and User Equipment (UE), work together to reduce the process load on each side and calculate the parameters used in decision-making phase. finally, UE decides to start or deny the HO process based on her velocity, current TTT, Reference Signal Received Quality (RSRQ), and Signal to Interference Plus Noise Ratio (SINR). Also, this model is not memory bounded and with only few bytes of memory the user's exact location can be estimated.

Finally, as presented in Figure 3, the proposed algorithm manages to handle the HO process while keeping the HoFR rate close to 0. This is done with keeping the TTT amount in less than 0.2s which will end in a seamless connection. Comparisons with similar HO algorithms [1] revealed that other algorithms cannot be used in scenarios between LTE-A and WiGig networks with the goals of time-efficiency and low data loss during a HO process. High (or constantly increasing) TTT values of other algorithms make them inappropriate to be used for managing the HO processes for these scenarios and in most cases HoFR could not be controlled or reduced by employing the methods used by them.

REFERENCES

[1] S. R. Niya, B. Stiller, "Design and Evaluation of a Time Efficient Vertical Handoff Algorithm between LTE-A and IEEE 802.11ad Wireless Networks," IFI Tecnical Report No.2018.03, Zürich, Switzerland, Tech. Rep., April 2018. [Online]. Available: https://files.ifi.uzh.ch/CSG/staff/Rafati/Handoff-IFI-2018.03.pdf

[2] H. Peng, K. Moriwaki, Y. Suegara, "Macro-Controlled Beam Database-Based Beamforming Protocol for LTE-WiGig Aggregation in Millimeter-Wave Heterogeneous Networks," in *IEEE 83rd Vehicular Technology Conference (VTC Spring)*, May 2016, pp. 1–6.

[3] J. Xu, Y. Zhao, X. Zhu, "Mobility Model Based Handover Algorithm in LTE-Advanced," in *10th International Conference on Natural Computation (ICNC)*, Aug 2014, pp. 230–234.

Poster: WebMaDa 2.0 - Automated Handling of User Requests

Corinna Schmitt, Dominik Bünzli, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
[schmitt|stiller]@ifi.uzh.ch, dominik.buenzli@uzh.ch

***Abstract*—Today users want to monitor their networks remotely and adjust privileges immediately. Addressing the first request is not a big problem anymore, because many applications offer such solutions by default (e.g., via a special app to be installed or a browser-based solution). The immediate privilege handling is the challenge nowadays, because usually a global administrator in the background needs to be included in the workflow. This is required, because he is the only person who has the full overview of the tool the network is included. In general this is a nice idea, but introduces delays to the privilege management depending of the number of networks linked to the system, in total. WebMaDa 2.0 overcomes this bottleneck by introducing an automated request handling solution to a web-based framework for monitoring sensor networks remotely as well as supporting privacy and immediate handling of privilege requests.**

***Keywords*- WebMaDa, automation**

I. Introduction

Today, many different devices are connected with each other building small networks that are part of the Internet of Things (IoT). Such networks are designed for individual solutions specialized for a specific purpose (e.g., environmental monitoring, health monitoring). Devices used show heterogeneity concerning hardware and software and are linked to a specialized solution allowing analysis and visualization of data collected. This itself is nothing really new within the IoT community. But the requests of users and network owners changed over time towards (1) mobility support, (2) ownership and controlling of data, as well as (3) updating granted privileges immediately.

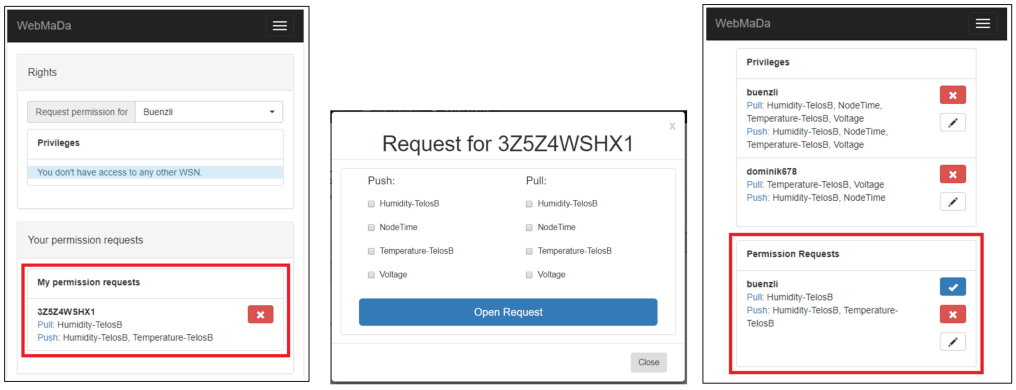
Many specific solutions are in place addressing the mobility request installing a special application on the mobile device. In general this is a good solution, but these solutions usually have special requirements to the operating system of the device and can exhaust the device quickly when running. The later can be overcome by integrating energy saving solutions, but still the applications require much memory of the device. To overcome this, web-based solutions are thought of beeing most suitable, because they only require Internet access and a browser installation on the device. Fortunately, both can be considered to be available by default on mobile devices. Furthermore, the code base only has to be updated in one place, thus reducing the cost for maintenance. The urge for control and ownership of the collected data is manifesting itself more and more in the minds of users. This is due to increased media coverage of data abuse caused by data leaks and the possibility of having data analyzed and visualized by third-party providers. Together with this situation comes the users' request to update granted privileges to manage access to the data collected. This is challenging, because access granted to applications can hardly be revoked or updated immediately if at all. Thus, the call for solutions supporting data and access control immediately arise. The aforementioned three issues (1)-(3) are addressed by WebMaDa, a Web-based Management and Data Handling Framework for sensor networks. The development started in 2014 with a basic support of mobile access to owned sensor networks allowing visualization of collected data in a flexible and hardware independent manner [2]. In 2016 WebMaDa received an update addressing the general request of fine-grained access management and pulling data in emergency cases [3]. The drawback was that each request (e.g., create networks, access to foreign networks, to view or pull data) required interaction of a global administrator introducing delay into the system. This drawback has now been solved in WebMaDa 2.0 [1] by automating the request handling within the system allowing immediate handling without the involvement of a global administrator. At the same time, the request for privacy and controlling data access is respected as every action that affects access rights is logged in the database.

In Section II, the main design decisions taken are presented leading towards the implemented WebMaDa 2.0 solution. Section III summarizes the new features of WebMaDa 2.0 highlighting the benefits and practical issues, as well as giving a hint to future improvements.

II. Design and Implementation

In order to handle any request received immediately an automated solution is required. This solution must support (1) user creation, (2) access request to foreign networks, and (3) password reset. Furthermore, for addressing privacy and controlling of the data (4) transparency must be assured by including a detailed logging system into the infrastructure.

Addressing the first three requests an automated mailing solution was integrated into WebMaDa 2.0. If a new user wants to use WebMaDa he needs to register by filling out the registration form. By submitting the form, the user creates an invitation request that is stored in the database. At the



(a) Requestor view (b) For for requesting access to data (c) Network owner's view of pending request

Figure 1: Graphical views during data access request

same time, the administrators receive a notification that a new request has been created. If the request is accepted by the administrators the user will receive an invitation which can be used to complete the registration. Otherwise, a message will be sent informing that the request has been rejected. After this, the user is able to create new networks or request access to foreign networks when not yet having permission as shown in Figure 1a. The latter is done by creating a permission request filling out a form (cf. Figure 1b). The form is received by the backend. Here a mapping between the selected network by the unique WSN Identifier (ID) and the stored owner address is performed resulting in mailing the request to him. The owner receives a mail with the request and a personalized link to handle it within WebMaDa (cf. Figure 1c). The owner can now grant the access, update the request or deny it. In return, the requester receives the result andl a log entry is created in the backend's database addressing the transparency request. Same procedure is followed if after time the network owner updates granted privileges. In case a registered WebMaDa user looses the password, a request can be placed via the corresponding form. The filled in data of the form is then compared to the logged entries in the database. If the check fails, no action is performed as not to provide a single bit of information wheter a user exists or not. Otherwise, the user receives a link to reset the password. In order to ensure transparency, the updating of a password also triggers the creation of a log entry.

III. Summary, Practical Issues, and Benefits

WebMaDa 2.0 supports the original functionality developed in 2014 and 2016. This is extended by an automated mailing solution to handle incoming requests immediately and, thus, reducing delays in the system each time an administrator interaction was required in earlier versions.
The designed and implemented solution is user-friendly due to its intuitive design in the graphical environment including easy understandable instruction to conclude the workflow (e.g., request data access, register new user). All steps are following a global process starting with a form that need to be filled out with respective information required, checkup with stored information if applicable, and updating database with new information (e.g., new user information, new networks, granted/updated/revoked privileges). In order to address the general privacy request of users the administrator is only involved when new users are registered or an existing WebMaDa user should become administrator of WebMaDa for the case the original administrator needs a representative. Addressing the transparency concerns of users, any changes are logged within the database with required information (e.g., timestamp, what was done and by whom). All this logging information can only be accessed by the network owner or the global administrator.
Looking from a practical perspective all user requests are addressed within WebMaDa without having drawbacks on performance of WebMaDa assuming several networks hosted at the same time. Due to the fact that WebMaDa 2.0 is still web-based, no new special requirements to the mobile device exist and the solution is still hardware and software independent.
Further developments are conceivable with regard to session timeout similar to banking systems, two-way authentication besides mailing using SMS, and further flexibility in visualizing data collected.

References

[1] D. Buenzli, "Efficient and User-friendly Handling of Access Requests in WebMaDa," Bachelor Thesis, Communication Systems Group, Department of Informatics, University of Zurich, Zurich, Switzerland, Jan. 2018.

[2] M. Keller, "Design and Implementation of a Mobile App to Access and Manage Wireless Sensor Networks," Master Thesis, Communication Systems Group, Department of Informatics, University of Zurich, Zurich, Switzerland, Nov. 2014.

[3] C. Schmitt, C. Anliker, and B. Stiller, "Pull Support for IoT Applications Using Mobile Access Framework WebMaDa," in *IEEE 3rd World Forum on Internet of Things (WF-IoT)*. New York, NY, USA: IEEE, Dec. 2016, pp. 377–382.