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

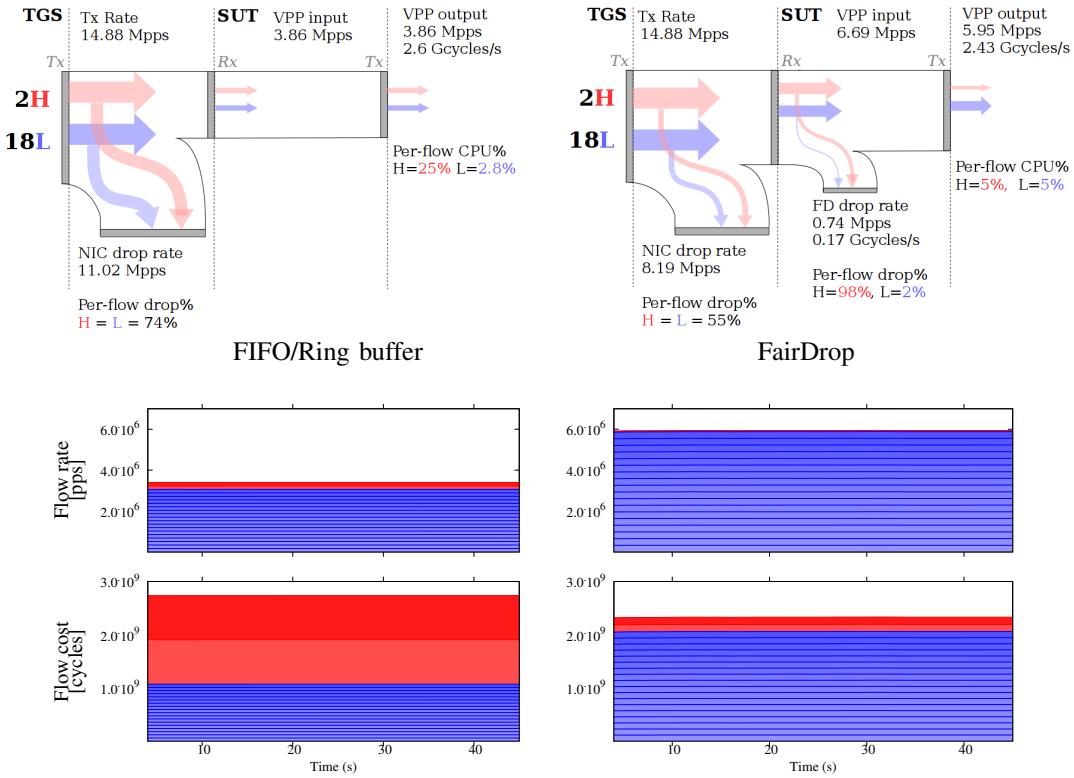

Fig. 1. Illustration of Classical (left column) vs FairDrop (right column) operations. Top part reports sankey diagrams of the rate at the Traffic Generator (TGS) and at the System Under Test (SUT). Lower part depicts the time evolution of the flow rate (in Mpps, middle) and the flow ccost (in cycles, 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/.