Performance Evaluation of Fast Startup Congestion Control Schemes
(Work in Progress)

Michael Scharf

Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart, Germany
michael.scharf@ikr.uni-stuttgart.de

Abstract. The Transmission Control Protocol (TCP) uses the Slow-Start mechanism at the beginning of a connection and after idle times. The Slow-Start delays the transport of data in particular if the round-trip time is large, which is undesirable for interactive applications. In order to speed up transfers, several alternatives have been proposed recently. This paper evaluates the performance and robustness of new fast startup congestion control schemes. We compare both end-to-end approaches as well as protocols that rely on additional feedback from the routers, using implementations in the Linux stack. Both testbed measurements and simulation studies quantify the potential performance improvement, the risk of packet loss, and the benefits of additional router support. Our results, which are also partly verified analytically, reveal that end-to-end fast startup mechanisms would not cause too much performance degradation if they are selectively used and carefully tuned. Additional router support would improve the fairness at the cost of a higher complexity.

Keywords: Congestion Control, TCP, Slow-Start, Quick-Start.

1 Introduction

Most Internet applications use the Transmission Control Protocol (TCP) for reliable, best effort transport. TCP uses congestion control [1] to adapt to the available bandwidth on the path. Still, after the connection setup or after idle periods it is difficult to determine an appropriate sending rate. TCP's congestion control uses the Slow-Start heuristic in these cases. This mechanism works well in the Internet, but it may require many round-trip times (RTTs) until an appropriate sending rate is reached and thus significantly delay data delivery.

This raises the question whether a faster startup is possible. The design of startup procedures for new flows is one of the remaining open issues of the Internet congestion control [2]. Recently, several new fast startup mechanisms have been developed, which all modify the Slow-Start. One solution is the Quick-Start TCP extension [3,4], which allows higher initial sending rates if the routers along the path approve a corresponding request. Since this requires modifications in the network entities, Quick-Start cannot be used in today's Internet. As a

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 716–727, 2009.

potential alternative, Jump-Start [5] has been proposed. Jump-Start is a pure end-to-end mechanism. It uses rate pacing with a high initial sending rate, but reacts conservatively if the available bandwidth is exceeded. Further possibilities include e. g. increasing TCP's initial window beyond the value allowed by [6].

This paper compares several fast startup congestion control mechanisms. We study both end-to-end solutions, which only require modifications in the sender, as well as Quick-Start TCP, which is an example for a router-assisted approach. Unlike simulation studies such as [4,5], we implement the new fast startup mechanisms in the Linux kernel in order to obtain realistic results. In addition to testbed measurements, we also run simulations using the Network Simulation Cradle [7] for the user-space execution of real network stack code. To the best of our knowledge, this is the first comparative study of the new fast startup schemes and thereby complements our work on Quick-Start TCP [8,9].

The rest of this paper is structured as follows: Section 2 discusses the design space of fast startup congestion control and reviews selected proposals. In Section 3 we present an analytical model that is used to verify our experiments. Section 4 introduces our Linux implementations and our evaluation methodology. In Section 5 we compare benefits and risks of different fast startup schemes using analysis, simulations, and measurements. Section 6 concludes the paper.

2 Fast Startup Congestion Control

2.1 The TCP Slow-Start and Enhancements

The Slow-Start is one part of TCP's congestion control strategy. The basic idea is to start a transfer with a small congestion window and increase the window by one segment whenever an acknowledgment (ACK) arrives. This results in an exponential increase of the window, until the *Slow-Start Threshold* is reached and the *Congestion Avoidance* phase is entered. Originally, the initial window was one segment [1], but today a value of three segments is permitted by [6].

Several extension of the TCP congestion control, such as the Cubic algorithm [10], have been developed to improve the Congestion Avoidance phase in high-speed networks. Still, most TCP stacks use the original Slow-Start [1], which causes two problems: First, it can take a long time until the source can

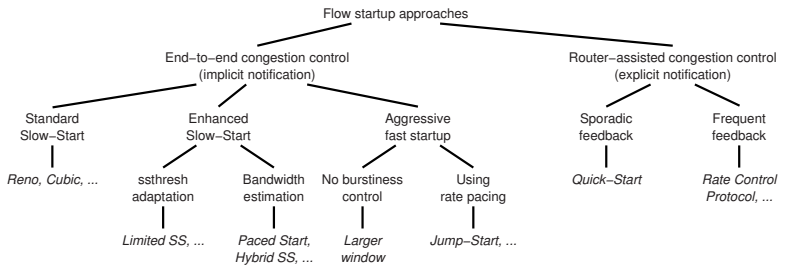

Fig. 1. Classification of flow startup principles

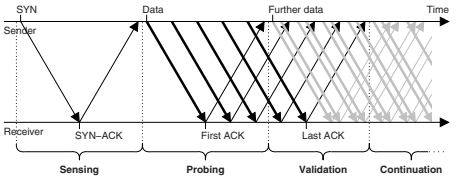

Fig. 2. Four phases of a fast startup

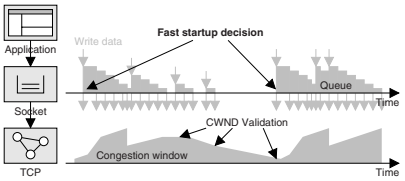

Fig. 3. Fast startup usage scenarios

fully utilize the available bandwidth on a path, in particular if the RTT is large. As a result, data transfers of interactive applications are unnecessarily delayed. Second, the exponential increase may be too aggressive ("Slow-Start overshoot") and cause multiple packet losses. An ideal congestion control should actually ensure that new flows converge quickly to their fair share of resources [2].

These issues are addressed by several improved startup approaches that are classified in Fig. 1. In order to overcome the overshooting problem, several enhancements have been developed, such as *Limited Slow-Start* [11]. Also, bandwidth estimation techniques have been proposed in order to determine the path capacity during the Slow-Start, e. g., with a *Paced Start* [12] or a *Hybrid Slow-Start* [13]. These enhancements still start with a small initial window.

2.2 Fast Startup Design Space

There are also several proposals how to speed up the Slow-Start and mitigate its performance limitations, which are surveyed e. g. in [3]. In general, a more intelligent end-to-end flow start startup mechanism could use some of the following information that is often available in end-systems: The round-trip time, cached state variables for this destination (e. g., a congestion manager), observable application communication characteristics (such as the amount of queued data in the socket), the local interface capacity, or application requirements.

Another option is to use explicit feedback from network entities along the path. Recently, several router-assisted congestion control mechanisms have been proposed. Potential solutions range from TCP enhancements, such as Quick-Start TCP [3], to completely new congestion control frameworks for a Future Internet. For instance, the Rate Control Protocol [14] suggests to uses some additional per-packet processing to speed up data transfers.

These fast startup mechanisms can be characterized by four phases that are shown in Fig. 2: During the *sensing* phase, some path characteristics are determined, potentially using explicit network feedback. Then the sender starts to send data (*probing*). The *validation* phase starts when corresponding ACKs arrive. The sender can then determine whether the initial choice was reasonable. Finally, the sender switches to the continuous congestion control (*continuation*), typically after the last ACK for the initially sent data has been received.

As depicted in Fig. 3, Slow-Starts do not only occur at the beginning of a connection, but also after longer idle periods, if the congestion window validation [15] is triggered. In this case fast *restart* mechanisms can be applied.

Table 1. Comparison of the considered fast startup schemes

	Quick-Start	**Jump-Start**	**More-Start**	**Initial-Start**
Sensing	Router feedback	—	—	—
Probing	Approved rate (rate pacing)	Play out app. data (rate pacing)	Use given rate (rate pacing)	Large window (no pacing)
Validation	Observe loss	Count retransm.	Count retransm.	unmodified
Continuation	Revert ofter loss	Adapt after loss	Adapt after loss	unmodified

2.3 Overview of Selected Fast Startup Approaches

In the following, we consider four different fast startup mechanisms:

Quick-Start TCP [3] is an experimental TCP extension that uses explicit router feedback. With Quick-Start, end-systems can ask for a high initial sending rate, e. g., during the three-way handshake. If all routers along the path approve the request, data transfers can start with this high rate. Further details about the Quick-Start mechanism and its implications can be found in [4,8,9].

Recently, Jump-Start [5] has been proposed as an end-to-end fast startup mechanism. The basic idea is to play out the queued application data during the first RTT using rate pacing. Thus, the more data is available, the higher is the initial rate. Jump-Start risks to start with a too large data rate. In order to cope with overshooting, the TCP retransmissions are counted during a validation phase. At the end of the loss recovery, the congestion window is adjusted to roughly half of the successfully transmitted segments. The authors of [5] argue that Jump-Start is not overly aggressive since most of today's TCP connections only transmit few data and would thus start with a rather small rate.

The principles of Quick-Start and Jump-Start can also be combined to a new end-to-end startup variant: Similar to Quick-Start, the applications could explicitly choose a "reasonable" initial rate, e. g., by considering application requirements or a congestion manager. Unlike Quick-Start, this rate is just used without asking routers for approval, in combination with a careful reaction to packet loss like Jump-Start. We label this combination "More-Start".

As a reference, we also consider the trivial mechanism of just increasing the initial congestion window to a larger value, without any further modifications of the TCP congestion control (Initial-Start). Table 1 highlights the differences between the four schemes and their functions during the four startup phases.

3 Speedup Analysis

3.1 Analytical Slow-Start Model

In order to verify our simulation and measurement results we use an analytical Slow-Start model developed by Bodamer (see [8]). As shown in Fig. 4, we model an end-to-end path by its TCP path capacity $R = \frac{L}{MTU} \cdot r$ and its minimum round-trip time τ. Therein, r is the data rate at IP layer, $MTU = 1500\,\text{byte}$ is the maximum transmission unit, and L is the maximum segment size (MSS).

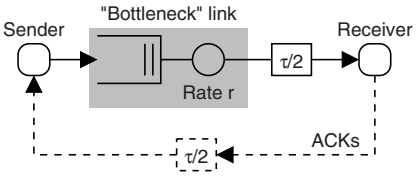

Fig. 4. Simplified path model

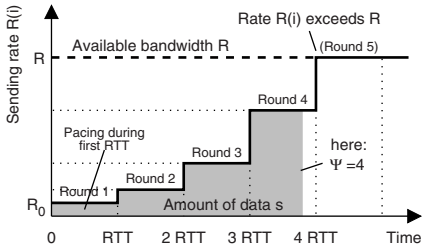

Fig. 5. Fast startup with rate pacing

Two further important parameters are the initial congestion window w and the number of segments b that a receiver acknowledges in one segment. As detailed in [8], the transfer time $\Gamma_{\mathrm{SS}}(s)$ of a certain amount of data s in Slow-Start is

$$\Gamma_{\mathrm{SS}}(s) = \frac{s}{R} + \left(\tau + \frac{L}{R}\right) \cdot \psi - \frac{L \cdot w}{R} \frac{\gamma^{\psi} - 1}{\gamma - 1}. \tag{1}$$

Eq. (1) neglects packet loss and limitations by the receive window or a small initial Slow-Start Threshold. It also does not include unidirectional latencies or connection setup delays. ψ is the index of the last Slow-Start round that is completely used, and $\gamma = 1 + \frac{1}{b}$. In [8] the index ψ is calculated as

$$\psi = \max(\min(\underbrace{\left\lceil \log_{\gamma}\left(\tfrac{1}{w}\left(\tfrac{R\cdot\tau}{L}+1\right)\right)\right\rceil}_{\text{Window exceeds BDP}}, \underbrace{\left\lceil \log_{\gamma}\left(\left\lceil \tfrac{s}{L}\right\rceil \tfrac{\gamma-1}{w}+1\right)-1\right\rceil}_{\text{Transfer completed early}}), 0). \tag{2}$$

From this model follows $T_{\mathrm{SS}}(s) = \Gamma_{\mathrm{SS}}(s) + \tau$ as minimum response time of a client-server application. The Linux network stack can be characterized by $L = 1448$ and $b = 1$, if the *Quick-ACK* mechanism [16] is active.

3.2 New Model for Rate Paced Fast Startup Schemes

The fast startup mechanisms considered in this paper use rate pacing during the first RTT. This results in a different behavior that can be modeled as follows: If the initial sending rate R_0 is smaller than the end-to-end available bandwidth R, and if the sender continues in Slow-Start, the data rate is increased by a factor γ every RTT, resulting in *rounds* as depicted in Fig. 5. During round i, data is continuously sent with rate $R(i) = R_0 \cdot \gamma^{i-1}$. The maximum amount of data that can be sent when round i is completed is $M(i) = R_0 \cdot \tau \cdot \frac{\gamma^i - 1}{\gamma - 1}$. The transfer time is then

$$\Gamma_{\mathrm{Paced}}(s) = \tau \cdot \Psi + \frac{s - M(\Psi)}{\min\left(R, R_0 \cdot \gamma^{\Psi}\right)}. \tag{3}$$

Similar to Eq. (2), the index Ψ of the last round with a rate $R(i) < R$ is

$$\Psi = \max(\min(\underbrace{\left\lceil \log_{\gamma}\left(\tfrac{R}{R_0}\right)\right\rceil}_{\text{Rate exceeds } R}, \underbrace{\left\lceil \log_{\gamma}\left(\tfrac{s\cdot(\gamma-1)}{R_0\cdot\tau}+1\right)-1\right\rceil}_{\text{Transfer completed early}}), 0). \tag{4}$$

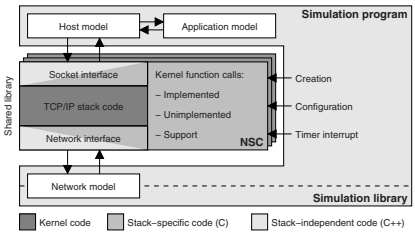

Fig. 6. Real code simulations with NSC

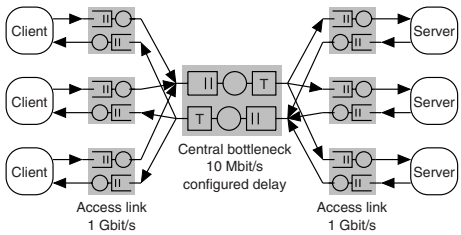

Fig. 7. Dumb-bell simulation topology

The minimum server response time for an approved Quick-Start rate q can be derived from Eq. (3) as $T_{\mathrm{QS}}(s) = T_{\mathrm{Paced}}(s) + \tau$ with $R_0 = \frac{L}{MTU} \cdot q$. As described in [3], there is only a limited number of valid values for q. It must be emphasized that Eq. (3) does not apply if the Slow-Start Threshold is adapted after the validation phase (cf. [9]). In case of Jump-Start, the initial rate R_0 depends on the amount of queued application data. If a threshold u is used, $T_{\mathrm{JS}}(s)$ can be obtained from (3) with $R_0 = \frac{\min(s,u)}{\tau}$. In the following we use $u = 64\,\mathrm{KiB}$. For Mean-Start, $T_{\mathrm{MS}}(s)$ can be determined similar to T_{QS}, but without limitations for the granularity for q. The minimum server response time $T_{\mathrm{IS}}(s)$ of Initial-Start follows directly from Eq. (1) with a larger value for w.

4 Implementation and Evaluation Methodology

4.1 Linux Network Stack Implementations

In order to compare the different fast startup schemes, we have implemented Quick-Start, Jump-Start, More-Start and Initial-Start in the Linux network stack. Our Quick-Start TCP implementation is described in [9]. Compared to Quick-Start, our implementations of end-to-end fast startup schemes are comparatively simple, since only TCP code is affected. Still, the realization of rate pacing in Jump-Start and More-Start causes some complexity, as it requires timers and an additional state machine. These aspects, as well as potential interactions with flow control, are addressed in [9]. Our Quick-Start patch requires more than 2000 additional lines of kernel code, while e. g. Jump-Start only needs several hundred lines. But it is not sufficient to modify only the congestion control implementation. For Jump-Start, the socket processing workflow must also be adapted in order to determine the amount of queued application data.

4.2 Simulation and Measurement Methodology

In our measurements we use computers with Ubuntu/Fedora systems and Linux kernel 2.6.24. We always activate the non-standard-compliant *Quick-ACK* mechanism [16] to make our results reproducible and independent of the activation heuristics of the kernel. This implies $b = 1$, i. e., in our studies the Slow-Start may be significantly faster than a standard-compliant TCP stack with $b = 2$.

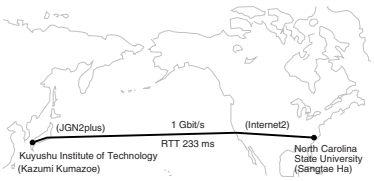

Fig. 8. Network path in experiments

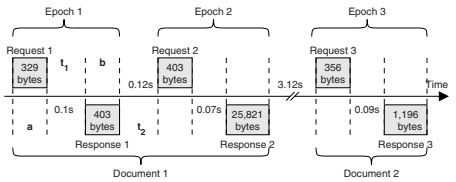

Fig. 9. Illustration of the a-t_1-b-t_2 model

We use the default stack configuration and the "Cubic" high-speed congestion control [10], which is Linux's default choice. The socket buffer sizes have been increased to 8 Mbyte in order to avoid limitations by the TCP flow control.

We also simulate with the Linux network stack in order to perform realistic, controlled studies with a larger number of entities. Our simulation tool uses the Network Simulation Cradle (NSC) version 3.0 [7] for the user-space execution of kernel code. The NSC architecture and its integration into our simulation tool is illustrated in Fig. 6. The corresponding wrappers and tools are documented in [17,18]. Our simulations are performed with kernel version 2.6.18, both without and with our fast startup patches.

In the simulations we use the classic dumb-bell topology [19] with N client-server pairs, a central bottleneck with $r = 10$ Mbit/s, a drop-tail buffer, and configurable delays (see Fig. 7). Unless stated otherwise, the buffer length is $B = 50$ packets and the RTT is $\tau = 200$ ms. In our local testbed, the client and server applications run on computers that are interconnected by 10 Mbit/s Ethernet segments. The Linux network emulation "NetEm" enforces latencies. Furthermore, we also perform some real-world experiments over a long-distance high-speed path, using the experimental infrastructure shown in Fig. 8.

For workload modeling we follow the approach of [20], i. e., the characteristics of client-server applications are described by traces of requests and responses as shown in Fig. 9. As simplest model we use fixed-sized small requests and responses. Furthermore, we study scenarios with a given downlink load, which is realized by scheduling request-response vectors over persistent TCP connections. The response length is Pareto distributed (mean $m = 125$ or 10 kbyte, shape factor $\alpha = 1.1$, cutoff at 10 MB), and the inter-arrival time is assumed to be exponential. We also replay Internet traffic traces [21] in a dumb-bell topology with nine different RTTs, as recommended in [19]. Our main performance metric is the server response time, which corresponds to the duration of epochs in Fig. 9.

5 Performance Comparison

5.1 Fundamental Startup Behavior

As a first step, we compare the Slow-Start (SS) to Quick-Start (QS), Jump-Start (JS), More-Start (MS), and Initial-Start (IS) in a very simple simulation setup: Figure 10 shows downlink traces for one client and one server ($N = 1$), with a

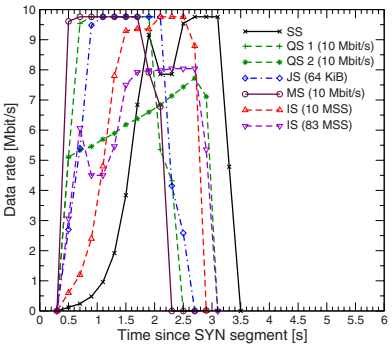

Fig. 10. Traces of a single startup (sampling time granularity 200 ms)

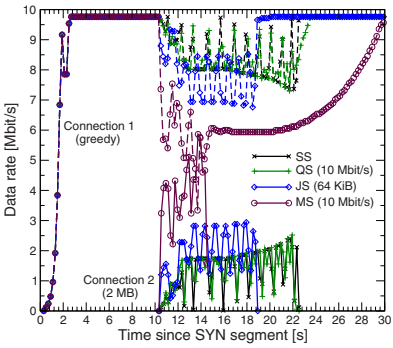

Fig. 11. Startup of a new flow against a long-lived competing TCP connection

request of 100 B and a response of 2 MB. Here, the Slow-Start needs over 1 s until the path is fully utilized. All fast startup mechanism are better in this scenario.

The largest approvable Quick-Start rate is $q = 5.12$ Mbit/s. If such a request is performed during the connection setup, and if it is approved, the data transfer can start immediately with this rate. After finishing the probing phase, there are different options [9]: The sender can either continue in Slow-Start ("QS 1") or adapt the Slow-Start Threshold to the Quick-Start window (about 83 segments) and continue in Congestion Avoidance ("QS 2"). The latter variant is more careful, but it may result in longer delays and is not further considered here.

Our Jump-Start implementation plays out up to $u = 64$ KiB during the first RTT and is thus starts slightly slower. In Fig. 10 the best performance would be achieved if the sender knew approximately the available bandwidth of $r = 10$ Mbit/s and used it (More-Start). Not shown is that a higher initial rate would, of course, cause packet loss, and the resulting completion time would be similar to Slow-Start. Finally, increasing the initial window to $w = 10$ would also work in this scenario. However, any initial value larger than the bottleneck buffer size will cause severe packet loss, resulting in no significant improvement.

A bottleneck link might already be occupied by existing flows. In order to study the convergence behavior in this case, a similar simulation scenario has been set up with $N = 2$ clients and servers and a start offset of 10 s. Figure 11 shows selected examples of the resulting flow startup. Three observations can be made: First, with Slow-Start, a short flow is unable to reach a rate of 5 Mbit/s, which would be its fair share. This long convergence is an inherent aspect of Slow-Start. Second, Quick-Start does not improve the convergence, if the routers use an admission control strategy that prevents over-commitment (cf. [8]). Due to lack of free capacity, the Quick-Start request is denied at the bottleneck, and the second connection therefore falls back to Slow-Start. Third, a fast startup e. g. with Jump-Start or More-Start improves the performance of the new flow at cost of the already established connection. It is an open and controversial question whether such an aggressive behavior of short flows is fair, nor not [2].

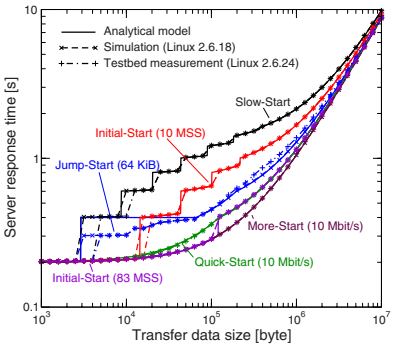

Fig. 12. Maximum possible speedup for path capacity 10 Mbit/s and RTT 200 ms

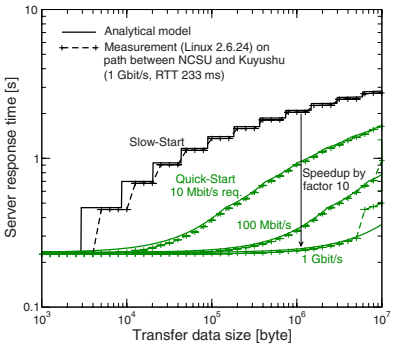

Fig. 13. Speedup on the high-speed path with different Quick-Start request rates

5.2 Speedup Compared to Slow-Start

The performance benefit of fast startup schemes depends on the amount of data s, the available bandwidth r, and the RTT τ, as determined in Section 3. Figure 12 compares the analytical values for the response times T_{SS}, T_{QS}, T_{JS}, and T_{IS} with the simulation and testbed measurement results for a large buffer ($B = 1000$ packets). The graph for standard TCP reveals the typical steps of the Slow-Start. As to be expected [4,8], a fast startup can improve the response time in particular for mid-sized response sizes. Figure 12 also reveals that model and experiments slightly differ in case of Jump-Start: If a perfect rate pacing was used, the response time for transfers up to $s = u = 64\,\mathrm{KiB}$ would be $T_{\mathrm{JS}} = \tau$. However, our real implementation only uses a limited number of timers (cf. [9]) and therefore sends faster if $s \ll u$. Figure 13 presents the results of a similar experiment with Quick-Start on a long-distance path ($r \approx 1\,\mathrm{Gbit/s}$). Due to lack of full Quick-Start router support, this study assumes that requests up to that rate are indeed approved. In such a high-speed environment, a fast startup mechanism can achieve substantial transfer time speedups up to a factor of 10.

Figure 14 studies the scenario of several connections sharing the bottleneck. The results have been obtained by simulating the exchange of requests and responses over $N = 50$ persistent TCP connections between 50 client-server pairs. In this case, Slow-Start is also used if the congestion window validation [15] is triggered. Obviously, when the load ρ at the bottleneck increases, the response times get larger. A lower bound can be determined by integrating (1) and (3) over the response size distribution. Figure 14 reveals some important differences between the different schemes: First, as the load increases, the Quick-Start and Slow-Start performance is similar because of the Quick-Start admission control. Quick-Start also needs one RTT for the signaling, which reduces the speedup. An Initial-Start with $w = 83\,\mathrm{MSS}$ again turns out to be detrimental in combination with a bottleneck buffer of $B = 50$ packets. Significant overshooting also occurs when one just starts with full link speed (More-Start). An interesting result is that Jump-Start can keep the response time at a low level even for high load.

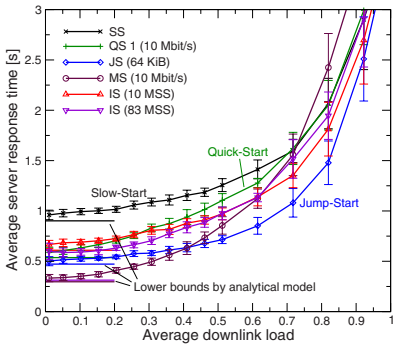

Fig. 14. Impact of the bottleneck load (Pareto distr. responses $m = 125\,\text{kB}$)

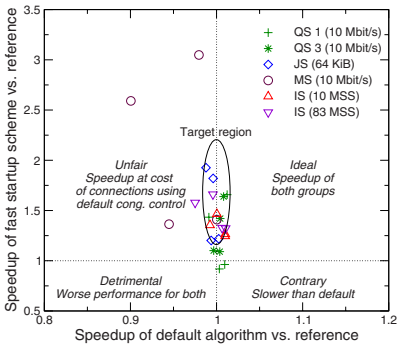

Fig. 15. Unfairness between competing default and fast startup connections

5.3 Fairness and Risk of Packet Loss

Any fast startup mechanism can result in fairness problems and risks to cause packet loss. Still, our experiments show that this may not be a severe problem if a sophisticated fast startup is used. Fig. 15 shows the result of an experiment similar to Fig. 14, except that there are now $N/2$ unmodified network stacks ("default") and $N/2$ that use a fast startup. Simulations are performed with two different traffic characteristics ($m_1 = 125\,\text{kB}$, $N_1 = 50$ and $m_2 = 10\,\text{kB}$, $N_2 = 400$) and two different loads ($\rho_1 \approx 0.05$, $\rho_1 \approx 0.3$). The x- and y-axis in Fig. 15 represent the resulting response times of the two groups. The reference value is the case that all end-systems use Slow-Start. In this representation, the interaction between the default and the fast startup mechanisms can be classified into four different cases. In the best case ("target region"), the fast startup schemes speed up their own transfers without significantly slowing down connections that use the default TCP congestion control. According to our results, both Quick-Start and Jump-Start are rather fair. The other variants can significantly affect the performance of connections that use an unmodified network stack.

5.4 Performance Impact for Typical Internet Workloads

The overall performance benefit of fast startup congestion control is difficult to quantify. Many TCP-based applications often transfer small objects that fit in today's initial congestion window, and most Internet RTTs are small, too. In these cases the existing Slow-Start performs reasonably well. This can also be observed in Fig. 16, which were obtained by replaying traffic traces [21] among $N = 450$ client-server pairs using realistic RTT distributions between 4 ms and 200 ms (cf. [19]). The traces are scheduled so that the downlink load of bottleneck is $\rho \approx 0.3$. If Jump-Start were used by all entities, epoch completion times that are of the order of several hundred milliseconds can be improved, but the overall benefit is not large. Using Quick-Start by default ("QS 1") even performs worse on such a moderately loaded link, since many requests get denied.

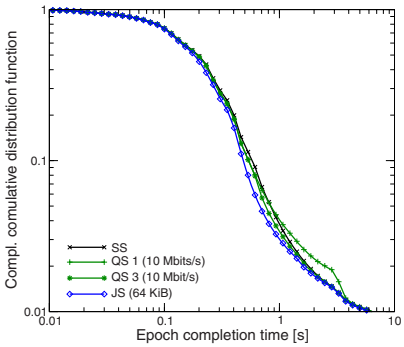

Fig. 16. Performance with traffic traces from [21] for a downlink load of 0.3

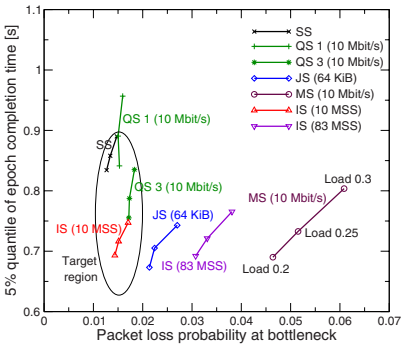

Fig. 17. Tradeoff between speedup and packet loss in the same setup

An improvement is possible if Quick-Start is only selectively enabled, e. g., for data transfers $> 10\,\mathrm{kB}$ ("QS 3"). The results for the other fast startup mechanisms, which are omitted, are similar to the Jump-Start graph.

Fig. 17 reports packet loss statistics in the same setup and relates them to the 5 %-quantile of the epoch completion time. The quantile is one possibility to quantify the speedup of selected communication patterns. With default stacks, the packet loss probability is of the order of 1 % in the given scenario. Quick-Start and Jump-Start, as well as a limited increased initial window, only moderately increase packet loss, while the other variants are more aggressive. These results again indicate that both Quick-Start and Jump-Start are not overly harmful, with Jump-Start being much simpler to implement and to deploy.

6 Conclusion and Future Work

New fast startup congestion control approaches aim at replacing the TCP Slow-Start. This papers studies the performance of four different mechanisms by analytical models, by simulation, and by measurements. We consider the Quick-Start TCP extension, the Jump-Start proposal, a new combination of both, and the simplest approach, i. e., just to increase TCP's initial congestion window. All schemes are implemented in the Linux stack. According to our results, Jump-Start performs well, even though it causes some unfairness to competing flows using Slow-Start. Of all schemes, Quick-Start is the most conservative one, but the required router support raises several unsolved issues. In contrast, the other more aggressive alternatives could cause harm. Specifically, it is not an option just to significantly increase the initial window without rate pacing. Also, a promising solution is to enable fast startup only in selected cases and for applications that can indeed benefit, but this would require additional intelligence in the network stack. Future studies could address some algorithmic details, in particular the response to packet loss. Further work is also needed to understand the overall implications of using fast startup congestion control on Internet scale.

Acknowledgments

This work is partially funded by the German Research Foundation (DFG) within the SFB 627. The author thanks the Kuyushu Institute of Technology and the North Carolina State University for providing access to their infrastructure.

References

1. Jacobson, V.: Congestion avoidance and control. In: Proc. ACM SIGCOMM 1988, pp. 314–329 (August 1988)
2. Welzl, M., Papadimitriou, D., Scharf, M., Briscoe, B.: Open research issues in Internet congestion control. IRTF Internet Draft, work in progress (August 2008)
3. Floyd, S., Allman, M., Jain, A., Sarolahti, P.: Quick-Start for TCP and IP. IETF RFC 4782 (experimental) (January 2007)
4. Sarolahti, P., Allman, M., Floyd, S.: Determining an appropriate sending rate over an underutilized network path. Computer Networks 51(7), 1815–1832 (2007)
5. Liu, D., Allman, M., Jin, S., Wang, L.: Congestion control without a startup phase. In: Proc. PFLDnet 2007 (February 2007)
6. Allman, M., Floyd, S., Partridge, C.: Increasing TCP's initial window. IETF RFC 3390 (proposed standard) (October 2002)
7. Jansen, S.: Simulation with real world network stacks. In: Proc. Winter Simulation Conference (2005)
8. Scharf, M.: Performance analysis of the quick-start TCP extension. In: Proc. IEEE Broadnets (September 2007)
9. Scharf, M., Strotbek, H.: Performance evaluation of quick-start TCP with a Linux kernel implementation. In: Das, A., Pung, H.K., Lee, F.B.S., Wong, L.W.C. (eds.) NETWORKING 2008. LNCS, vol. 4982, pp. 703–714. Springer, Heidelberg (2008)
10. Rhee, I., Xu, L.: Cubic: A new TCP-friendly high-speed TCP variant. In: Proc. PFLDnet 2005 (February 2005)
11. Floyd, S.: Limited slow-start for TCP with large congestion windows. IETF RFC 3742 (experimental) (March 2004)
12. Hu, N., Steenkiste, P.: Improving TCP startup performance using active measurements: algorithm and evaluation. In: Proc. IEEE ICNP, pp. 107–118 (November 2003)
13. Ha, S., Rhee, I.: Hybrid slow start for high-bandwidth and long-distance networks. In: Proc. PFLDnet 2008 (March 2008)
14. Dukkipati, N.: Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly. Ph.D thesis, Stanford University (October 2007)
15. Handley, M., Padhye, J., Floyd, S.: TCP congestion window validation. IETF RFC 2861 (experimental) (June 2000)
16. Sarolahti, P., Kuznetsov, A.: Congestion control in Linux TCP. In: Proc. USENIX Annual Technical Conference (June 2002)
17. Zeeh, C.: Integration of the Linux-TCP/IP Protocol Stack into an Event-Driven Simulation Environment. Diploma thesis, University of Stuttgart, IKR (2006)
18. Proebster, M.: Performance Evaluation of Congestion Control with Explicit Router Signaling. Diploma thesis (in German), University of Stuttgart, IKR (2008)
19. Andrew, L., et al.: Towards a common TCP evaluation suite. In: Proc. PFLDnet 2008 (March 2008)
20. Weigle, M.C., Adurthi, P., Hernández-Campos, F., Jeffay, K., Smith, F.D.: Tmix: A tool for generating realistic TCP application workloads in ns-2. ACM SIGCOMM Computer Communication Review 36(3), 65–76 (2006)
21. Web site: WAN in Lab. (2008), `http://wil.cs.caltech.edu/suite/TrafficTraces.php`