# Quick-Start TCP: From Theory to Practice

Michael Scharf, Simon Hauger, Jochen Kögel
Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
{michael.scharf, simon.hauger, jochen.koegel}@ikr.uni-stuttgart.de

*Abstract*—**Router-assisted congestion control schemes are considered to be one potential solution to improve the transport protocol performance in high-speed wide-area networks. The experimental Quick-Start TCP extension uses explicit router feedback to avoid the standard Slow-Start mechanism. This allows a TCP connection to almost immediately utilize a high-speed path and can significantly speed up broadband interactive applications. Similar to other recently proposed new congestion control approaches, Quick-Start requires modifications both in TCP connection endpoints as well as in routers.**

**While the theoretical performance benefit of Quick-Start is evident, its implementation and usage in practice has rarely been addressed so far. This paper gives an overview of our Quick-Start implementation efforts and real-world experiments. First, we report from our experiences with an implementation in the Linux protocol stack, which has been extended to offer full Quick-Start functionality. Another implementation of the new router functions on a network processor shows that fast path processing of Quick-Start is technically feasible at very high line speeds. Second, our measurements show how Quick-Start performs in practice, and they quantify the benefit compared to the existing TCP congestion control. Finally, we determine the performance costs of using Quick-Start. Our measurements suggest that the Quick-Start processing overhead is very moderate.**

## I. Introduction

The Transmission Control Protocol (TCP) is the default transport protocol for reliable, elastic traffic in the Internet. The TCP congestion control continuously probes the available resources on the path. However, the standard algorithms are not well suited for paths with a large bandwidth-delay product that require large congestion windows to be fully utilized. New high-speed TCP extensions have been proposed to overcome this problem, but there are still situations that are challenging for any pure end-to-end congestion control [1]. One example is the beginning of a connection when any information about the path characteristics is missing. Most existing TCP variants use V. Jacobson's Slow-Start heuristic [2] in this case.

One solution to overcome this lack of information is explicit feedback from routers. Quick-Start TCP [3] is an experimental TCP extension that defines such a feedback: It allows end-systems to determine an initial sending rate in cooperation with the routers on the path, in particular at the beginning of data transfers. Senders can then start to send with a much higher data rate than the Slow-Start would allow. Technically, this is realized by a new Internet Protocol (IP) option that the routers have to process. This results in significant deployment challenges. However, Quick-Start is still a lightweight, evolutionary approach compared to other explicit signaling schemes that completely substitute the TCP congestion control.

A detailed description of the Quick-Start mechanism is given in [3]. Also, simulation studies such as [4], [5] have demonstrated that Quick-Start could significantly speed up best effort data transfers in underutilized high-speed networks. There are several other alternatives to overcome the TCP Slow-Start, which are surveyed e. g. in [6]. Most of them are less robust and potentially much more aggressive than Quick-Start.

The implementation of Quick-Start in existing TCP/IP protocol stacks, as well as experiments under real constraints, have hardly been addressed so far. This makes it difficult to comprehensively evaluate the usefulness of Quick-Start. And it also prevents a detailed comparison with other related router-assisted congestion control schemes, such as the eXplicit Control Protocol (XCP) or the Rate Control Protocol (RCP), for which implementations do exist [7]–[9].

This paper addresses the question how Quick-Start support can be realized in practice and summarizes our experiences from real-world tests. We have developed a Linux kernel patch that adds Quick-Start support to the TCP/IP stack, which is also described in [10]. Furthermore, we have implemented the Quick-Start router function in an Intel IXP network processor in order to demonstrate that the required processing of IP options is possible in high-speed networks with full line speed [11]. In this paper, we analyze and compare both implementations. We study the benefits and costs of the Quick-Start mechanism in several testbed setups, and we identify issues that are challenging in the practical use.

The rest of this paper is structured as follows: Section II gives an overview of our Linux implementation and addresses some of the lessons learnt, extending the documentation in [10]. In Section III, we briefly show how Quick-Start support can be added to a network processor, summarizing the results of [11]. Section IV presents several testbed measurements with both implementations. The results prove that Quick-Start TCP can significantly speed up applications compared to the traditional congestion control. Section V then studies the processing overhead caused by the Quick-Start support. Finally, Section VI concludes the paper.

## II. Linux Implementation of Quick-Start

### A. Required Functions

The Quick-Start TCP extension requires both modifications in the connection endpoints and in the routers along the path.

*Endpoint modifications*: In order to support Quick-Start, end-systems have first to be able to handle the new IP and TCP options. A Quick-Start originator must be able to add

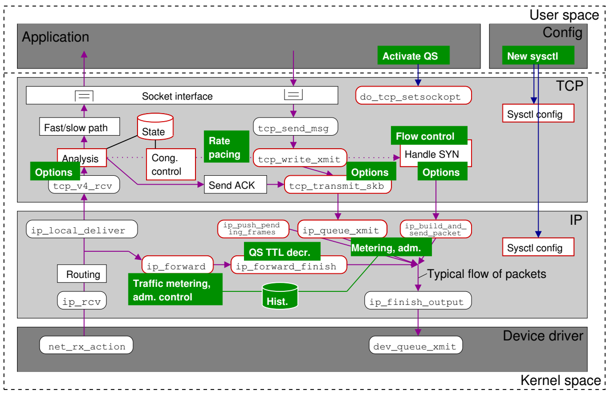

Fig. 1. Realization of Quick-Start in the Linux networking stack (cf. [10])

a "Quick-Start request" option to the header of IP packets, which includes the desired sending rate. If this option arrives at the receiving end-system, it must be processed and the contained data rate must be echoed back by a TCP option ("Quick-Start response"). Second, the handling of the TCP congestion window must be changed if a Quick-Start request is successful. Since standard TCP is not able to send with a defined data rate, a rate pacing mechanism has to be added. Finally, further changes are required in several other TCP functions, for instance concerning the flow control [12].

*Router support*: Quick-Start requires three additional functions in the routers: (1) Routers have to determine the capacity of the outgoing links and to keep track of their utilization. (2) The new IP option must be processed, which requires a couple of simple operations. And (3), an admission control logic is needed that decides which rate to grant to requests. None of these functions requires per flow state in routers.

### B. Implementation Overview

We have developed a patch for the Linux kernel (currently based on version 2.6.24) that adds the Quick-Start functions to the IPv4 and TCP layer. An illustration of the required changes in the Linux networking stack [13] is given in Fig. 1. The IP layer is mainly extended by the processing of the new IP option for outgoing and forwarded packets. It implements the "target algorithm" admission control [4]. The actual Quick-Start logic is realized by modifications of the TCP implementation. We have added several heuristics that can automatically issue a Quick-Start request during the connection setup, or when a connection has been idle for a longer time. Alternatively, an application can explicitly activate Quick-Start by setting a new socket option. The important parameters can be configured by additional "sysctl" variables, "ioctl" calls, or modified userspace tools. For instance, the following command enables Quick-Start on a certain interface with a maximum grantable data rate of 100 Mbit/s:

```
  $ ifconfig eth0 capacity 100000000
```

Further implementation details can be found in [10].

A TCP modification such as Quick-Start affects only a small part of the TCP/IP stack. Our patch adds or modifies less than 2000 lines of code. However, minor changes are required at many different places, which results in some ugly code and interface extensions. For instance, the Linux stack is not well prepared for sporadically added IP options and the thereby required reduction of the TCP maximum segment size.

An open issue that requires further work is the question how a Quick-Start enabled router can automatically determine the capacity of potential bottleneck links. The Linux kernel offers interfaces to network device drivers that can provide some information, but there is no "one-fits-all" solution. This issue is inherent to most router-assisted congestion control schemes. Another unsolved question is what data rate Quick-Start shall request for. Currently, we assume that either the applications can specify a reasonable rate, or that a system-wide default value can be defined (e. g., the local interface capacity).

## III. Quick-Start Router in a Network Processor

### A. High-Speed Router Realization

Any router-assisted congestion control scheme requires support in the core routers, which massively use hardware components for fast path packet switching in order to achieve line rates of multiple Gbit/s. Network processors are an increasingly popular technology for realizing flexible routers. They usually consist of several specialized processors, a general purpose processor, and fast memory and line interfaces. The specialized processors are optimized for packet processing tasks. Network processors are also extensively used for investigating new experimental Internet protocols (e. g., [9]).

### B. Implementation Overview

In order to study Quick-Start in high-speed hardware components we have implemented the required router functions in a network processor. Our solution is based on the Intel IXP 2400 network processor, which features eight specialized processors and one general purpose processor. We extended the standard router application provided by the manufacturer's software framework. As illustrated in Fig. 2, the router functions are divided into several functional pipeline stages.

Current routers typically process packets with IP options in the slow path without hardware support. The slow path has only a limited processing capacity and may significantly delay packets. A slow path processing of the Quick-Start IP options would not be an optimal solution if they were widely used, even if the options are typically only set in few packets.

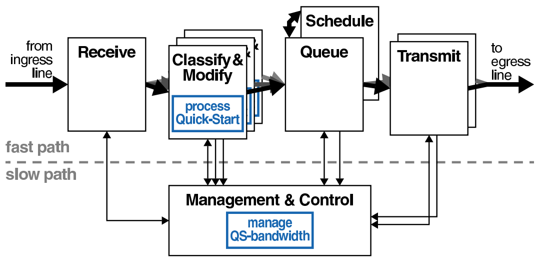

Fig. 2. Network processor based router with Quick-Start (QS) support

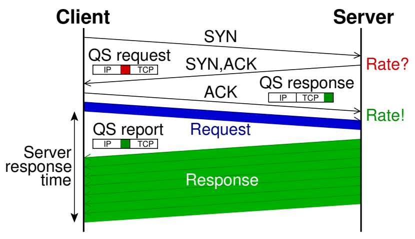

Fig. 3. Client-server communication with Quick-Start

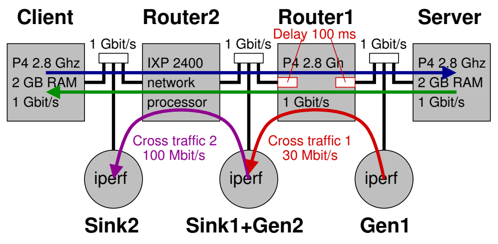

Fig. 5. Testbed with the network processor and several computers

Therefore, our implementation processes the new options in the fast path. This means that changes were mainly required in the microcode for the specialized processors. The processing of the Quick-Start requests was added to the pipeline stage that performs the IP processing (e. g., route lookup, classification, modification). Furthermore, the management of the available spare capacity on the output links was added to the functionality of the general purpose processor. A more detailed description of our implementation, as well as resulting issues concerning the parallel processing on multiple processors, can be found in [11].

## IV. Quick-Start Performance Measurements

### A. Measurement Methodology

We tested our Quick-Start implementations with a client-server communication as illustrated in Fig. 3. Both client and server are straightforward C programs under Linux that use standard socket calls. The client typically sends a small request (100 byte), and the server's response has a configurable size (from 1 kbyte to 100 Mbyte). If Quick-Start support is enabled, the kernel-internal heuristics in the server can automatically issue a request during the connection setup. In some tests the client application also sends further requests after some idle time, reusing the established connection. If the idle time is long, the congestion window validation [14] can significantly reduce the congestion window. When the kernel detects such a situation, it can automatically send a new request.

Client and server were interconnected in two different network setups: First, the two personal computers were connected back-to-back with a 100 Mbit/s Ethernet link. Second, we used a more complex network topology with two routers as shown in Fig. 5. One router is realized by the network processor. In both setups, the Linux “NetEm” mechanism was used to enforce a minimum round-trip time (RTT) of either 100 ms or 200 ms. In some cases, we also added constant-bitrate UDP cross-traffic to the links, using the “iperf” tool. The Linux machines utilize the “Cubic” congestion control variant [15], which is the default choice in the used Ubuntu 7.04 distribution. Most other IP and TCP stack parameters were set to their default values, except for increased socket buffers (8 Mbyte) and disabled connection statistics caching. During most measurements we captured “tcpdump” traces at the client. Furthermore, we measured the server response time as an application performance metric (cf. Fig. 3).

### B. Verification of the Quick-Start Endpoint Functions

Figures 4 and 6 show traces of the beginning of a download from the server, both with the Slow-Start and with the Quick-Start mechanism. In both diagrams, the data rates have been obtained by averaging the amount of received data per RTT. The server either sends a Quick-Start requests of 81.92 Mbit/s or asks for a lower value of 5.12 Mbit/s only. Furthermore, we distinguish between two different strategies to handle the Slow-Start threshold (“ssthresh”): After a successful Quick-Start request, we either leave the threshold unchanged, i. e., it remains at its default initial value, which is equal to the maximum integer value in recent Linux kernels. Alternatively, the threshold is reduced to the approved Quick-Start window, i. e., the sender enters the congestion avoidance phase after a Quick-Start. Both variants are possible according to [3].

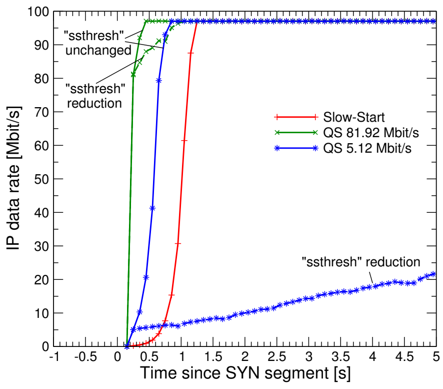

Fig. 4. Traces of Slow-Start vs. Quick-Start (100 ms RTT)

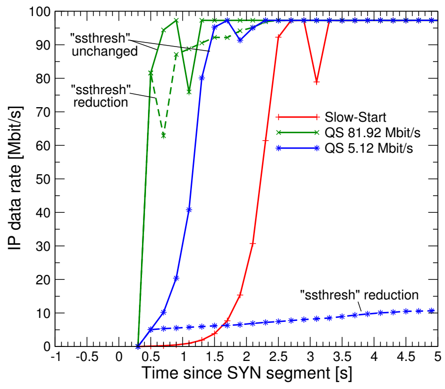

Fig. 6. Traces of Slow-Start vs. Quick-Start (200 ms RTT)

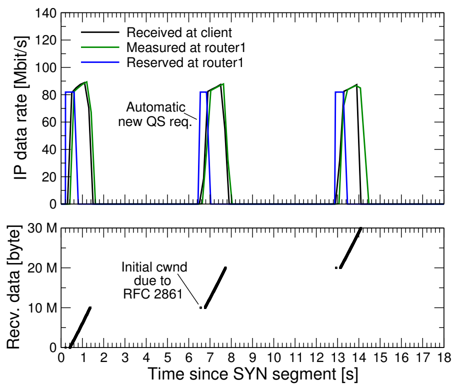

Fig. 7. Communication over a persistent connection without cross-traffic

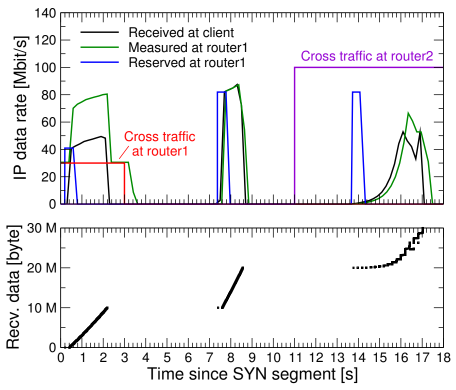

Fig. 8. Communication over a persistent connection with cross-traffic

The traces in Figures 4 and 6 confirm that a sufficiently high Quick-Start request allows the connection to almost immediately utilize the link capacity of 100 Mbit/s. In contrast, the Slow-Start forces a sender to send initially with a much lower rate, and it requires at least one second to achieve the full link speed. But the traces also reveal that Quick-Start does not necessarily result in faster data transfers: If the requested rate is smaller than the available bandwidth, and if "ssthresh" is adapted according to the approved Quick-Start bandwidth, the sender enters the congestion avoidance phase too early and increases the data rate rather slowly. Then the sending rate grows only slowly beyond the requested Quick-Start rate of 81.92 Mbit/s or 5.12 Mbit/s, respectively. If a "Reno" congestion control [3] was used instead of "Cubic", it would even take minutes to reach full link speed. This suggests that not reducing the threshold after a successful Quick-Start is a superior strategy. Therefore, our default choice in the following is not to reduce "ssthresh", unless stated otherwise.

## C. *Tests of the Quick-Start Router Functions*

In order to validate the router functions both in the Linux implementation as well in the network processor based router, we studied the topology sketched in Fig. 5. As application example, the client here downloads three times a data volume of 10 Mbyte over a persistent TCP connection, with 5 s idle time. The entities were configured with a Quick-Start admission threshold [3] of 100 Mbit/s. In the routers, the currently carried traffic and the Quick-Start request history has been extracted from (optional) system log messages.

Fig. 7 shows the resulting communication as observed both by the client and router 1 when there is no other competing traffic. The traces in the upper part of Fig. 7 confirm that, as to be expected, all three Quick-Start requests get approved. All three downloads start almost immediately with the requested rate of 81.92 Mbit/s. Afterwards, the rate increases only slowly because the "ssthresh" reduction strategy has been used in this setup. The router also precisely estimates the resulting traffic.

The lower part of Fig. 7 lists the sequence numbers received by the client. This gives some insight into the impact of the idle time: As congestion window validation is enabled, the sender can start the second and third transfer only with the small default initial window of three segments. The Quick-Start request is piggy-packed to first packet, i. e., the high-speed data transfer starts once the corresponding acknowledgement with the Quick-Start response has arrived after one RTT.

In order to illustrate the impact of the Quick-Start admission control, in the following the experiment is repeated, but now with additional inelastic traffic on the output links of the routers. As shown in Fig. 5, a cross traffic of 30 Mbit/s interferes with the first download, and during the third download there is a background load of 100 Mbit/s. Fig. 8 reveals that router 1 indeed measures a carried traffic of 30 Mbit/s when the first Quick-Start request arrives. Due to the admission threshold of 100 Mbit/s, it can only approve 40.96 Mbit/s. As a consequence, the data transfer starts with this rate. In contrast, the output link of router 1 is idle when the third Quick-Start request arrives. Therefore, it approves and reserves 81.92 Mbit/s. However, now the admission control at router 2 denies the request, since the load on its outgoing interface exceeds the Quick-Start admission threshold. As a consequence, the third download uses a Slow-Start, and router 1 spuriously reserves Quick-Start capacity for some time (two slots of 200 ms).

## D. *Parametrization of the Admission Control*

Our admission control and traffic metering functions operate on fixed time intervals ("slots"). The utilized bandwidth $u$ on an interface is calculated for slots of duration $D$. Furthermore, a history of approved Quick-Start requests is stored for a certain number of slots $H$, in order to avoid an over-granting of capacity. If there are many requests, the parameters $D$ and $H$ have a considerable impact on the capacity that can be granted. Both must be chosen carefully according to the maximum expected RTT [5] . In theory, if $q$ is the Quick-Start admission threshold of the "target algorithm" [4], the maximum capacity that can be granted per second is $\frac{q-u}{D \cdot H}$. This relationship is confirmed by the measurement results in Fig. 9. The diagram has been obtained by flooding a Linux-based router with many Quick-Start requests (200 req/s for 5.12 Mbit/s each). Evidently, the larger the slot $D$ duration and/or the length of the reservation history $H$, the more conservative is the admission control. In the default configuration of our implementation we use a rather careful setting of $D = 1000$ ms and $H = 2$.

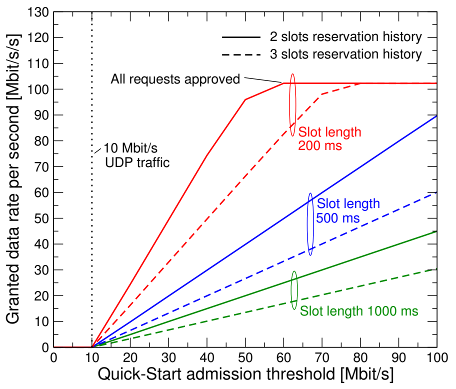

Fig. 9. Stress test of the admission control in the Linux stack

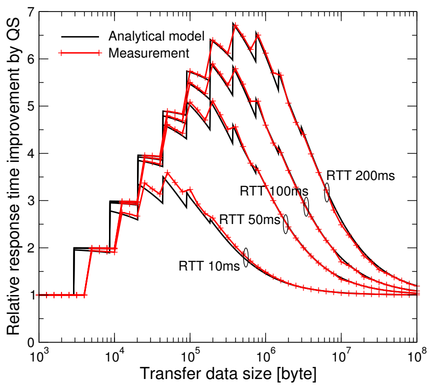

Fig. 10. Maximum Quick-Start benefit (100 Mbit/s link, 81.92 Mbit/s req.)

## *E. Speedup of Interactive Applications*

One of the most interesting questions is how much Quick-Start can improve application performance. Simulations [4] and analytical models [5] predict that the transfer times of moderate-sized files can be improved by several hundred percent. This can be verified with our implementation: Fig. 10 quantifies the relative performance improvement by Quick-Start for a client-server communication, i. e., the server response time with Slow-Start divided by the one with Quick-Start. The measurement results, which correspond to the analytical model in [5], reveal a significant speedup for data transfers between 10 kB and 10 Mbyte. In this range, the Slow-Start has a delaying effect. Smaller amounts of data can be sent out immediately with the initial window, and for long bulk data transfers the Slow-Start has only a marginal impact. Furthermore, the performance benefit significantly depends on the RTT. The larger the RTT, the higher the potential speedup.

We also performed experiments with real HTTP downloads, using a "lighttpd" web server and a modified "surge" load generator. The latter can issue requests using HTTP/1.0, HTTP/1.1, and HTTP/1.1+pipelining. Of course, the performance benefit of Quick-Start depends a lot on the characteristics of the Web pages. For small pages of the order of 10 kbyte, there is hardly any performance benefit. However, recent measurement studies such as [16] reveal that there are emerging new Web applications that frequently query objects with a size of 100 kbyte or more (high-resolution images, complex 3 D data, etc.). For object sizes of this order of magnitude, Quick-Start can be of significant benefit.

Table I compares the Web page loading durations for two Web models. On the one hand, we use the well-known "surge" model [17], where most objects are rather small. On the other hand, we use a customized model with larger page sizes (see also [10]). We assume a mean object size of 250 kbyte, which is consistent with some findings of [16]. The most significant improvement in Table I can be observed for HTTP/1.0 transfers and large object sizes. This is to be expected, since HTTP/1.0 transfers each object over a new connection with an initial Slow-Start. But Quick-Start can also considerably speed up the loading times in the other cases.

## V. Quick-Start Implementation Overhead

### *A. Impact of the Quick-Start Endpoint Functions*

Of course, the support of Quick-Start TCP comes at some cost, since it requires modifications both in the endpoints of a connection as well as in the routers on the path. Concerning the Linux end-system implementation, our experiments have revealed that Quick-Start adds only little processing overhead. All in all, the additional processing required by the Quick-Start functionality is simple compared to the complexity of the Linux TCP/IP stack as a whole. The main new functionality in the TCP stack is the rate pacing, but this function is only active for rather short periods of time. Thus, there must be many parallel TCP connections to cause a significant impact.

### *B. Impact of the Quick-Start Router Functions*

The key challenge of router-assisted congestion control is the additional packet processing that is required in routers. Fig. 11 and Table II give some insights into the impact of Quick-Start on the router performance, using workloads with different shares of packets that carry a Quick-Start request. Concerning the peak throughput, i. e., the maximum packet rate that can be transported without packet loss, our Quick-Start implementations do not suffer from a significant degradation compared to a corresponding router without Quick-Start support. As shown in Table II, the maximum packet rate of the network processor is about 3.9 Mpps without and with Quick-Start support. These packet rates have been measured for a workload with 50 byte sized IP packets, i. e., they correspond to a throughput of multiple Gbit/s with typical packet size

TABLE I
Comparison of the average Web page loading time with Slow-Start (SS) and Quick-Start(QS)

| HTTP variant | Small pages model SS | Small pages model QS | Large pages model SS | Large pages model QS |
|---|---|---|---|---|
| HTTP/1.0 | 0.97 s | 0.70 s | 2.97 s | 1.47 s |
| HTTP/1.1 | 0.65 s | 0.48 s | 1.49 s | 0.92 s |
| HTTP/1.1+p. | 0.62 s | 0.46 s | 1.36 s | 0.81 s |

*Setup:* 100 Mbit/s link, 200 ms RTT, 5.12 Mbit/s requests, 4 emulated users, 1h measurement time

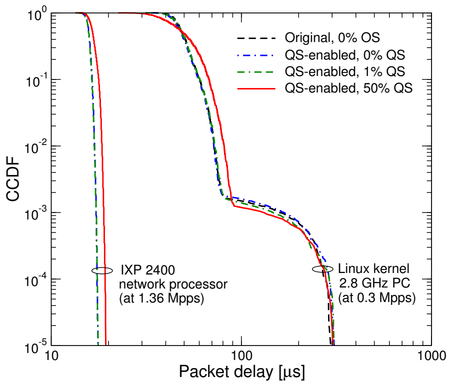

Fig. 11. Distribution function of the packet delays through a router

distributions. For very frequent Quick-Start requests there is a small decrease that can be attributed to synchronization issues among the fast path processing entities [11].

On a Linux PC with Intel Gbit/s Ethernet cards, the maximum achieved packet rate is about one order of magnitude smaller. If every second packet includes a Quick-Start option, the total system load of a router increases by about 15 %, mainly due to the recalculation of the random Quick-Start nonces. However, such an occurrence of Quick-Start options is only realistic in denial-of-service attacks. Interestingly, we observed that in some cases the increased system load results in a higher peak packet rate. This is probably a side effect caused by the handling of soft and hard interrupts in the kernel.

We also measured the delay that packets experience when traversing a router, using an Agilent J6800A network analyzer for workload generation and high-precision tracing. Both the mean values in Table II and the complementary cumulative distribution function (CCDF) in Fig. 11 reveal that the Quick-Start option processing causes an additional delay, but it is less than 1 $\mu$s. If the traffic estimation slot interval is short, i. e., less than 1 s, the delay can be slightly larger due to the frequent updates of the traffic statistics. Still, the impact on the total delay inside a router is neglectible. This result shows again that the overhead of Quick-Start is very small.

TABLE II
IMPLEMENTATION PERFORMANCE CHARACTERISTICS

| **Peak packet rate (50 byte size)** | **Linux PC, P4 2.8 GHz 1 Gbit/s Eth., PCI bus** | **IXP 2400 network processor** |
|---|---|---|
| Original, 0 % QS | 0.33 Mpps | 3.9 Mpps |
| QS-enabled, 0 % QS | 0.33 Mpps | 3.9 Mpps |
| QS-enabled, 1 % QS | 0.33 Mpps | 3.9 Mpps |
| QS-enabled, 50 % QS | 0.35 Mpps | 3.8 Mpps |
| **Mean packet delay (50 byte size)** | **Linux PC, P4 2.8 GHz 1 Gbit/s Eth., PCI bus at 0.3 Mpps** | **IXP 2400 network processor at 1.36 Mpps** |
| Original, 0 % QS | 50 $\mu$s | 16 $\mu$s |
| QS-enabled, 0 % QS | 50 $\mu$s | 16 $\mu$s |
| QS-enabled, 1 % QS | 50 $\mu$s | 16 $\mu$s |
| QS-enabled, 50 % QS | 51 $\mu$s | 17 $\mu$s |

## VI. CONCLUSION

Quick-Start is an experimental TCP extension that could improve TCP performance over long-distance networks, provided that the required protocol stack modifications get widely deployed. This paper gives an overview of our experiences with implementing Quick-Start, both in the TCP/IP stack of the Linux operating system as well as in a network processor. Our software and hardware-supported implementations allow to test Quick-Start in a wide range of scenarios. Our findings demonstrate, on the one hand, that Quick-Start would be a beneficial mechanism for broadband applications, in particular if they interactively exchange large amounts of data. On the other hand, our measurements suggest that Quick-Start support causes only little overhead in end-systems and routers.

However, there are still challenges that require further work. Open issues in our implementations include in particular the design of interfaces towards lower protocol layers and also towards applications. Another important aspect is to evaluate the benefits and costs of Quick-Start compared to other congestion control schemes using explicit router feedback.

## REFERENCES

[1] M. Welzl, D. Papadimitriou, and M. Scharf, "Open research issues in Internet congestion control," IRTF Internet Draft, work in progress, July 2007.

[2] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," RFC 2581 (proposed standard), Apr. 1999.

[3] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-Start for TCP and IP," IETF RFC 4782 (experimental), Jan. 2007.

[4] P. Sarolahti, M. Allman, and S. Floyd, "Determining an appropriate sending rate over an underutilized network path," *Computer Networks*, vol. 51, no. 7, pp. 1815–1832, May 2007.

[5] M. Scharf, "Performance analysis of the Quick-Start TCP extension," in *Proc. IEEE Broadnets*, Sept. 2007.

[6] D. Liu, M. Allman, S. Jin, and L. Wang, "Congestion control without a startup phase," in *Proc. PFLDnet2007*, Feb. 2007.

[7] A. Falk, T. Faber, E. Coe, A. Kapoor, and B. Braden, "Experimental measurements of the eXplicit Control Protocol," in *Proc. PFLDnet2004*, Feb. 2004.

[8] N. Dukkipati, G. Gibb, N. McKeown, and J. Zhu, "Building a RCP (Rate Control Protocol) test network," in *Proc. IEEE HOTI*, Aug. 2007.

[9] K. Nakauchi and K. Kobayashi, "An explicit router feedback framework for high bandwidth-delay product networks," *Computer Networks*, vol. 51, no. 7, pp. 1833–1846, 2007.

[10] M. Scharf and H. Strotbek, "Performance evaluation of Quick-Start TCP with a Linux kernel implementation," in *Proc. IFIP Networking 2008, Springer LNCS*, May 2008.

[11] S. Hauger, M. Scharf, J. Kögel, and C. Suriyajan, "Quick-Start and XCP on a network processor: Implementation issues and performance evaluation," submitted to IEEE HPSR 2008.

[12] M. Scharf, S. Floyd, and P. Sarolahti, "Avoiding interactions of Quick-Start TCP and flow control," IETF Internet Draft, work in progress, July 2007.

[13] K. Wehrle, F. Pählke, H. Ritter, D. Müller, and M. Bechler, *The Linux Networking Architecture*. Prentice Hall, 2005.

[14] M. Handley, J. Padhye, and S. Floyd, "TCP congestion window validation," IETF RFC 2861 (experimental), June 2000.

[15] I. Rhee and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," in *Proc. PFLDnet2005*, Feb. 2005.

[16] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann, "The new Web: Characterizing AJAX traffic," in *Proc. 9th International Conference on Passive and Active Network Measurement, Springer LNCS*, Apr. 2008.

[17] P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 1, pp. 151–160, 1998.