Universität Stuttgart

INSTITUT FÜR
KOMMUNIKATIONSNETZE
UND RECHNERSYSTEME
Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn

Copyright Notice

©2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

Institute of Communication Networks and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: +49-711-685-68026, Fax: +49-711-685-67983
email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

Performance Analysis of the Quick-Start TCP Extension

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

***Abstract*—Quick-Start is an experimental extension of the Transmission Control Protocol (TCP) that allows to speed up best effort data transfers. With Quick-Start, TCP hosts can request permission from the routers along a network path to send at a higher rate than allowed by the default TCP congestion control. The explicit router feedback avoids the time-consuming capacity probing by the TCP Slow-Start and is therefore particularly beneficial for underutilized paths with a high bandwidth-delay product. In this paper, the performance of the Quick-Start TCP extension is analyzed and the impact of router admission control strategies is studied. The main contribution is an analytical model that quantifies the improvement compared to default TCP Slow-Start. The model is validated by simulation results and by initial measurements with a Quick-Start implementation in the Linux operating system. Our results confirm that Quick-Start can significantly reduce the completion times of mid-sized data transfers.**

I. Introduction

The Transmission Control Protocol (TCP) is the default transport protocol for best effort Internet services. Since TCP is a pure end-to-end protocol, the connection endpoints have to estimate the path characteristics in order to adapt their sending rate. However, after connection setup, or e. g. after long idle periods, it is difficult to determine an appropriate rate due to lack of information. Traditionally, the TCP congestion control uses the "Slow-Start" heuristic to probe the available bandwidth in these cases, but this is a time-consuming process, since it can require many round-trip times to reach an appropriate sending rate.

The Quick-Start extension [1] addresses this issue by using explicit router feedback. It allows hosts to ask for an initial sending rate, e. g., during the TCP three-way handshake, and the routers along the path can approve, modify, or discard this request. If the Quick-Start request is approved, using a higher-than-default initial sending rate can significantly speed up high-speed best effort data transfers over paths with a large bandwidth-delay product, such as over long-distance broadband wide area networks, satellite or cellular links.

Even though Quick-Start is only a minor extension of the TCP congestion control, its deployment has various implications on the architecture and performance of IP networks. An initial evaluation of Quick-Start in [2] quantifies the performance benefits compared to standard TCP and discusses basic router strategies. Yet, many details and trade-offs are not discussed in detail there, and the results are based on simulations only. This paper derives analytical approximations for the performance of Quick-Start. To the best of our knowledge, this is the first analytical analysis of Quick-Start that quantifies the performance benefit and that approximates the impact of router admission control procedures. Although our model is an initial approach and does not consider all involved effects, it confirms some of the key findings of [2]. Furthermore, we report some measurement results that have been obtained from a new Quick-Start implementation in the Linux TCP/IP protocol stack, which has been developed in [3].

The rest of this paper is structured as follows: Section II gives an overview of the Quick-Start TCP extension and related work. In Section III, an analytical model is presented that quantifies how much Quick-Start can improve the completion time of data transfers with limited size, compared to the standard Slow-Start. Section V discusses design choices for routers that support the Quick-Start mechanism. In Section VI, different parametrizations for the router algorithms are studied analytically and by simulation. The results are verified by measurements with our Linux kernel Quick-Start implementation in Section VII. Finally, Section VIII concludes the paper.

II. The Quick-Start TCP Extension

A. Overview

Quick-Start is an experimental TCP extension standardized by the IETF [1]. With Quick-Start, TCP connection end-points can rapidly determine an allowed sending rate in cooperation with the routers on the path, in particular at the beginning of a data transfer. Thus, Quick-Start is basically a performance enhancement for elastic best effort transport over paths with significant free capacity.

Fig. 1 illustrates a Quick-Start request during TCP connection establishment: In order to indicate its desired sending rate, Host 1 adds a "Quick-Start request" option to the IP header. This option includes a coarse-grained specification of the target rate, encoded in 15 steps ranging from 80 kbit/s to 1.31 Gbit/s. The routers along the path can approve, modify, or disallow this rate request. Each router that supports the Quick-Start mechanism performs an admission control and reduces or discards the request if there is not enough bandwidth available.

If the request arrives at the destination Host 2, the granted rate is echoed back piggybacked as a TCP option ("Quick-Start response"). The originator can then detect whether all routers along the path support Quick-Start and whether all of them have explicitly approved the request. If not, the default

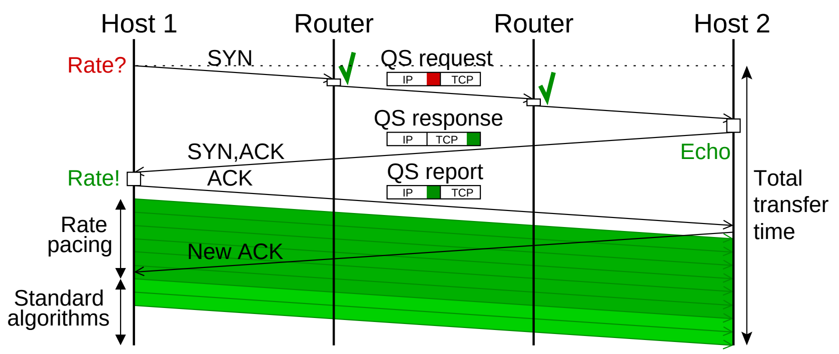

Fig. 1. Illustration of a Quick-Start request during the TCP three-way handshake

congestion control (i. e., TCP Slow-Start) is used to ensure backward compatibility.

If the Quick-Start request is successful, the originator can increase its congestion window to a value potentially much larger than the initial value allowed by [4]. Provided that Host 2 announces a sufficiently large receive window that does not restrict the sender, Host 1 can start to send with the approved rate, using a rate pacing mechanism (see Fig. 1). A special problem can arise if the receive window larger is larger than 65,535 byte, since window scaling is not allowed in segment with the "SYN" flag. A simple workaround is to send an additional empty acknowledgment after the "SYN,ACK", which announces the true receive window, or to modify the TCP receive window scaling mechanism [5]. After one round-trip time (RTT), the Quick-Start phase is completed and the default TCP congestion control mechanisms are used for the subsequent data transfer.

B. *Benefits and Open Issues of Quick-Start*

The Quick-Start extension can significantly enhance the TCP performance over paths with a high bandwidth-delay product, since it avoids the time-consuming Slow-Start. Simulation results in [2] reveal that the transfer times of moderate-sized files can be improved by several hundred percent (cf. Section IV-B). In addition to avoiding initial Slow-Starts, the Quick-Start mechanism could also be quite useful in the middle of data transfers, e. g., after longer idle periods, or in combination with link layer mobility triggers [6]. Quick-Start can thus complement other new high-speed congestion control mechanisms that target at environments with a large bandwidth-delay product (see e. g. [7]).

However, Quick-Start TCP, as any explicit router feedback approach, comes at some cost: The Quick-Start mechanism requires support by *all* routers along a path. As a consequence, it is difficult to deploy Quick-Start incrementally, and there are interworking problems e. g. with IP tunnels. Furthermore, existing middleboxes such as firewalls may drop packets carrying Quick-Start options. Quick-Start also causes processing overhead in routers, even though they do not have to store per-flow state. Routers also need some knowledge about link layer characteristics, which could require cross-layer information exchange, for example for links with variable bandwidth. Concerning security, Quick-Start includes protection mechanisms against misbehaving routers and receivers, but it is vulnerable to certain denial-of-service attacks.

There are also open research questions concerning the interaction with applications, for instance, when to trigger Quick-Start requests, and how to determine the data rate to request for. As discussed in [1], [2], reasonable choices could be made e. g. by knowing the last-mile link capacity. Furthermore, interactive applications could for instance determine the required bandwidth by taking into account the amount of data and/or a response time deadline.

As some of these issues require further research, [1] concludes that initial deployment of Quick-Start should be limited to controlled and trusted environments such as Intranets.

C. *Related Work*

The Quick-Start TCP extension is a lightweight, coarse-grained, in-band explicit router feedback mechanism that rapidly discovers the available bandwidth on a path. A similar router feedback is already realized by Explicit Congestion Notification (ECN) [8]. Unlike ECN, Quick-Start does not target at improving TCP behavior over congested links.

There are numerous other proposals to mitigate the performance limitation of TCP's Slow-Start. A comprehensive survey of this related work can be found in [1], [2]. Four principal approaches can be distinguished: (1) Schemes that apply bandwidth estimation techniques in order to determine the available bandwidth, (2) additional function blocks that share path capacity information, such as a "congestion manager", (3) explicit feedback from network elements along a path, and (4) proposals to use an arbitrarily high sending rate at the beginning of data transfers. The forth category, i. e., the idea to "stupidly" start sending with a potentially high data rate, has recently been proposed by [9] and [10]. However, such a sender behavior can result in severe congestion, and the impact on networks is not well understood so far. Compared to this, Quick-Start, which belongs to the third category, is a much more careful and conservative mechanism.

Ongoing research activities for future "clean slate" Internet architectures include completely new congestion control schemes that are based on more powerful explicit router feedback mechanisms. Such explicit notification can be achieved either with in-band signaling or with out-of-band signaling. For instance, the Explicit Control Protocol (XCP) [11], [12] uses fine-grained, in-band, per-packet feedback from routers in order to improve performance in networks with a high bandwidth-delay product. Compared to Quick-Start TCP, these proposals are more complex to realize and therefore not further considered in this paper.

As already mentioned, reference [2] examines the performance of the Quick-Start mechanism by simulation. Another recent study [13] analyzes the performance of Quick-Start over satellite links. It confirms significant benefits for the start-up behavior of streaming traffic with "TCP Friendly Rate Control". Simulations in [6] show that Quick-Start can be used to improve the TCP performance after vertical handovers. The analytical analysis in the following section extends and complements this initial work on Quick-Start performance.

III. Analytical Model

A. Analyzing Short TCP Connections

The performance improvement of Quick-Start can be quantified by analyzing the sojourn time of a given amount of data after connection setup. Two different factors may limit the performance of TCP in Slow-Start: First, the throughput can be affected by a maximum sending window, a limited Slow-Start threshold, a small receiver advertised window, or by packet loss. These cases have been analyzed e. g. in [14], [15]. Second, the TCP throughput is limited once it reaches the available bandwidth of the end-to-end path. While an optimized configuration can overcome window restrictions, the second case imposes a fundamental limitation. Models for the corresponding performance can be found for instance in [16], [17]. These models are extended in [18], where a closed-form expression for file transfer times is given. In the following, we briefly review some of the findings of the model in [18]. Based on this we then quantify the maximum Quick-Start performance improvement.

B. Model Assumptions

As shown in Fig. 2, we assume that the end-to-end path can be characterized by a bottleneck link with a TCP processing capacity $R = \frac{L}{MTU} \cdot r$ and a minimum round-trip time τ that incorporates all other path delays. r is the data rate for IP packets, MTU is the maximum transmission unit (assumed to be 1500 bytes), and L is the maximum segment size (1460 bytes). The bandwidth delay product, including the bottleneck link, thus follows as $P = \frac{R \cdot \tau}{L} + 1$. Note that P, as well as other variables, is counted in segments of size L. In the following, we also assume that the initial value of the Slow-Start threshold and the receive window are larger than P. This ensures that the throughput is not limited by an unfavorable TCP configuration.

The TCP behavior in Slow-Start can be modeled by "rounds" that start when the sender begins the transmission of a window of packets and that end when an acknowledgment (ACK) for one or more of these packets arrives [14]–[17]. In each round $i \geq 1$, the sender sends as many data segments as its congestion window $W(i)$ allows. With the so-called "delayed acknowledgments", the receiver typically sends an ACK for every $b = 2$ data segments (see e. g. [14]). Since the sender increases its congestion window by one segment per ACK, the congestion window at round $i+1$ follows as $W(i+1) \approx (1 + \frac{1}{b})\, W(i) = \gamma\, W(i)$ with $\gamma = 1 + \frac{1}{b}$. If the sender starts with an initial window w, the window size in round i, i. e., the maximum amount of data to be sent in that round, is $W(i) = w \cdot \gamma^{i-1}$. The amount of data transmitted in all rounds up to i can be approximated by a geometric series: $M(i) = \sum_{j=1}^{i} W(j) = w\, \frac{\gamma^i - 1}{\gamma - 1}$. For $MTU = 1500$ bytes, [4] allows a maximum initial window $w = 3$.

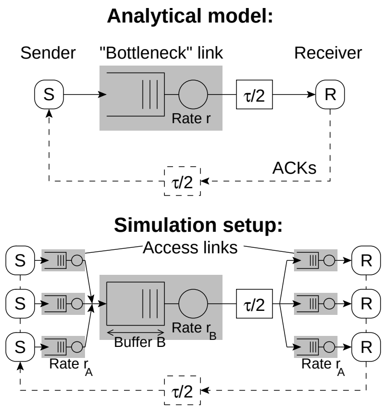

Fig. 2. Network models in the analysis and in the simulations

C. Transfer Times with Slow-Start

Given a certain amount of data, parts of it may be transferred in Slow-Start rounds. Once the sending window $W(i)$ exceeds the bandwidth delay product P, the throughput is mainly limited by the data rate of the path. Further assuming that the duration of a round is $\tau + \frac{L}{R}$ and independent of the window size, we can determine the Slow-Start transfer time $\Gamma_{\text{SS}}(s, R)$ of a given amount of data s as follows:

$$\Gamma_{\text{SS}}(s, R) = \frac{\tau}{2} + \underbrace{\left(\tau + \frac{L}{R}\right) \cdot \psi}_{\text{Delay by Slow-Start}} + \underbrace{\left(\frac{s - L \cdot M(\psi)}{R}\right)}_{\text{Transfer fully utilizing path}} . \quad (1)$$

In this expression, ψ is the index of the last Slow-Start round that is completely used by the sender. In order to calculate ψ, two cases have to be considered: First, the sending window may exceed the bandwidth-delay product P in round κ, i. e., $W(\kappa) \geq \frac{R \cdot \tau}{L} + 1$. Solving for κ gives:

$$\kappa = \left\lceil \log_\gamma \left(\frac{1}{w} \left(\frac{R \cdot \tau}{L} + 1 \right) \right) \right\rceil + 1 \; . \quad (2)$$

Second, short transfers may not arrive at this point. If the data is completely transferred in Slow-Start, the number of rounds ν follows from $M(\nu) \geq \lceil \frac{s}{L} \rceil$:

$$\nu = \left\lceil \log_\gamma \left(\left\lceil \frac{s}{L} \right\rceil \frac{\gamma - 1}{w} + 1 \right) \right\rceil \; . \quad (3)$$

By definition, the last complete round has the index

$$\psi = \min(\kappa, \nu) - 1. \quad (4)$$

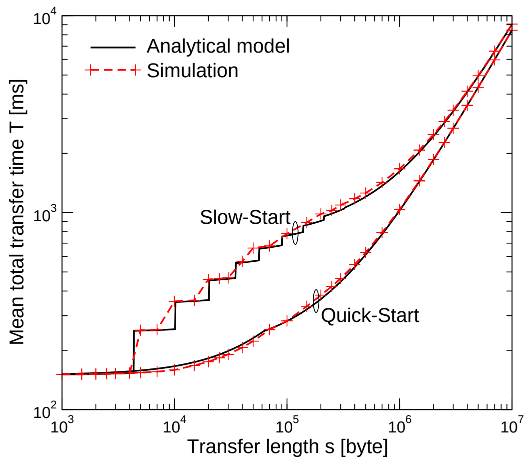

Fig. 3. Transfer times ($\tau = 100\,\text{ms}$)

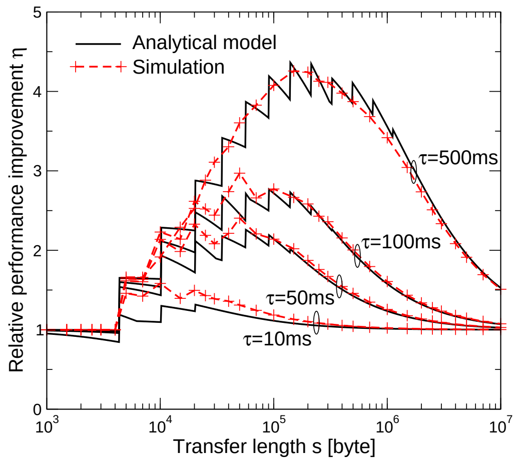

Fig. 4. Improvement by Quick-Start

D. Transfer Times with Quick-Start

With Quick-Start, the data transfer can immediately start with the data rate q granted by the routers along the path. The sender uses this data rate during one RTT; afterwards, the sending rate may be increased up to r by the standard TCP congestion control procedures. Assuming that the additional time to fully utilize the path is small, the transfer time can be approximated by

$$\Gamma_{\text{QS}}(s,R,Q) = \frac{\tau}{2} + \begin{cases} \frac{s}{Q} & \text{if} \quad s < Q \cdot \tau \\ \tau + \frac{s - Q \cdot \tau}{R} & \text{else} \end{cases}, \qquad (5)$$

where $Q = \frac{L}{MTU} \cdot q$ is the achieved TCP throughput during the rate pacing phase. This formula can be explained as follows: If $s < Q \cdot \tau$, all data is transferred in the rate phasing phase of maximum duration τ. If more data has to be transferred, the data transmission of the remainder is governed by the default TCP congestion control, which can potentially utilize the full available bandwidth r. In the best case $q \approx r$, i. e., if the requested Quick-Start rate is close to the link capacity, Eq. (5) can be simplified to $\Gamma_{\text{QS}}(s,R,Q) \approx \frac{\tau}{2} + \frac{s}{R}$. For $q \ll r$, Eq. (5) may underestimate the real transfer time, since the TCP congestion control may prevent the sender from immediately sending with r after having completed the rate pacing phase.

E. Total Transfer Times

For a data transfer from the connection originator to the server, one has to take into account an additional delay of τ for the three-way-handshake. The total transfer time is then

$$T = \tau + \Gamma, \qquad (6)$$

where Γ is either $\Gamma_{\text{SS}}(s,R)$ or $\Gamma_{\text{QS}}(s,R,Q)$. When multiple connections share the bottleneck link, the sojourn times Γ could roughly be approximated by extending this model using M/G/1 or M/G/r processor sharing approximations [16], [17].

IV. Performance Improvement of Quick-Start TCP

A. Simulation Methodology

In order to compare standard Slow-Start and Quick-Start TCP, simulations have been performed with the IKR simulation library, which implements a “NewReno” TCP congestion control [19]. In order to get an initial understanding of the performance, we use a simple dumbbell topology with one shared link, as depicted in Fig. 2. Unless otherwise mentioned, the data rate of the shared link is $r_{\text{B}} = 100\,\text{Mbps}$ with a drop-tail buffer of size $B = 75000\,\text{byte}$ (50 packets). The minimum round-trip time is $\tau = 100\,\text{ms}$. A variable number of senders and receivers is connected by access links with rate $r_{\text{A}} = 10\,\text{Mbps}$. The senders realize unidirectional short-lived data transfers by establishing separate TCP connections, thus roughly modeling HTTP/1.0-like communication.

The model in our simulation tools implements Quick-Start according to [1]. For simplicity, we assume in the following that the senders always try to use the Quick-Start mechanism during connection setup and request for a data rate of $q = 5.12\,\text{Mbps}$, which is the maximum value for the access links. We use a large maximum receiver window of 10 Mbytes, and the initial Slow-Start threshold is set to 10 Mbytes, too. These settings ensure that TCP performance is not restricted in our setup. As mentioned in [1], we adjust the Slow-Start threshold after the successful completion of a Quick-Start phase and set it to two times the allowed Quick-Start sending window. During a successful Quick-Start phase, packets are sent with a rate pacing granularity of 1 ms.

In Section VI, the mean transfer size is assumed to be $m = 250\,\text{kbyte}$. In addition to constant transfer sizes, we also use either Lognormal distributed sizes S with $f_S(x) = P[S = x] = \frac{1}{\sqrt{2\,\pi} \cdot \sigma \cdot x} \cdot \exp\left(-\frac{(\ln x - \mu)^2}{2\,\sigma^2}\right)$ for $x > 0$, where $\mu = 11.9292$ and $\sigma = 1.0$, or a Pareto distribution with shape factor $\alpha = 1.4$, i. e., $f_S(x) = P[S = x] = \frac{\alpha \cdot k^\alpha}{x^{\alpha+1}}$ for $x \geq k = \frac{\alpha - 1}{\alpha} m$. This workload model represents a simple interactive application with moderate-sized data transfers that could probably benefit most from Quick-Start. However, in

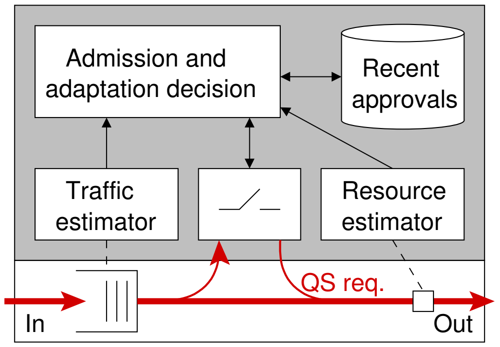

Fig. 5. Quick-Start router functions

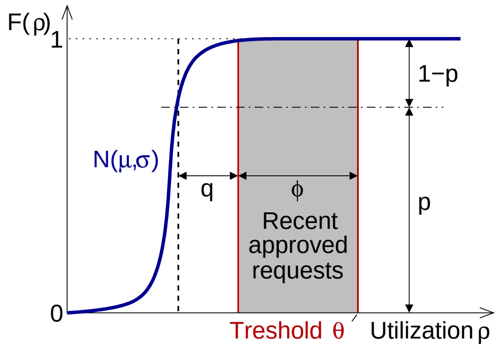

Fig. 6. Success probability derivation

reality, the communication pattern of Web applications may be more complex, even though Lognormal or Pareto distributions are predominant in the Web [20]. For simplicity, we assume Poisson arrivals of data transfers with mean inter-arrival time $\frac{1}{\lambda}$. This assumption is reasonable when data transfers are issued by a large number of users [17]. Each simulation run includes 10 batches, each with one million packets, and the results are presented with 95 % confidence intervals.

B. Comparison of Standard TCP and Quick-Start TCP

First, we study the maximum performance benefit that Quick-Start can achieve if the path is completely unloaded, i. e., when only one connection shares the bottleneck. As example we consider only one access link with rate $r_{\mathrm{A}} = 10\,\mathrm{Mbps}$, assuming Quick-Start requests for $q = 5.12\,\mathrm{Mbps}$.

Fig. 3 shows the total transfer time T as a function of the transfer size s, both for standard TCP and Quick-Start. Therein, the steps occur when a new Slow-Start round is required. Quick-Start significantly speeds up transfers in the range of 10 kbyte to 1 Mbyte. As expected, the graph has a small kink at $s \approx Q\tau = 64$ kbyte. Quick-Start is only of limited benefit for small sizes s, since transfers can be completed in just a few round-trip times anyway. Also, Quick-Start does not significantly improve long bulk data transfer, where the Slow-Start is only a transient phase.

In Fig. 4, the relative improvement $\eta = \frac{T_{\mathrm{SS}}(s,R_{\mathrm{A}})}{T_{\mathrm{QS}}(s,R_{\mathrm{A}})}$ of Quick-Start over standard Slow-Start is depicted for different latencies τ. Both our analytical analysis and the simulation results show that Quick-Start can improve transfer times for moderate-sized transfers by several 100 %, in particular if the network latency is high. Our analytical analysis thus confirms similar empirical findings in [2].

V. Router Support for Quick-Start

A. Required Router Functions

The support of Quick-Start requires additional functions in the routers, which are shown in Fig. 5: (1) Routers have to determine the capacity of the outgoing links and keep track of their utilization. (2) Routers must perform an admission control, i. e., decide whether to accept a Quick-Start request and which data rate to grant. And (3), information about recently approved requests must be stored.

Similar functions are known from measurement-based admission control for guaranteed Quality-of-Service (QoS) [21]. Yet, unlike such IntServ-like QoS mechanisms, routers supporting Quick-Start are not required to make any guaranteed reservations of bandwidth. In the following, we briefly evaluate the design space for the three building blocks, extending the discussion in [1], [2].

B. Estimation of Available Bandwidth

Quick-Start requires an online estimation of the available bandwidth on the egress interfaces. The capacity of router interfaces can be determined for instance from the corresponding network technology. If the capacity is not constant (e. g., on shared wireless links), this may require cross-layer information exchange. Nevertheless, given the coarse granularity of Quick-Start requests, some inaccuracy can be tolerated.

When the link capacity is known, the available bandwidth can be derived from measurements of the transferred data volume, either within certain time-slots, or with a window-based solution [22]. The measurement can be combined with low-pass filters, such as the exponentially weighted moving average (EWMA), forecasts, or trend-based approaches [23]. In [2], it is proposed to use an EWMA or a “peak utilization” estimator. However, parameterizing such smoothing filters is known to be a non-trivial problem [23]. In the following, we assume that rate measurements are performed in time intervals D, and we mainly focus on estimators without EWMA. A further promising alternative would be active bandwidth probing techniques, but they are not further studied in this paper.

C. Admission and Adaptation Decision

The admission control for Quick-Start requests tries to ensure that the link utilization stays within acceptable limits. [2] proposes the “target algorithm”, which sets an upper threshold θ on the link utilization ρ (see Fig. 6). Requests are approved if the resulting utilization ρ, augmented by the bandwidth from recent Quick-Start approvals ϕ, is less than θ. If a request exceeds θ, the permitted rate is reduced or the request is dropped. We label this strategy “with reduction”.

An alternative could be an admission control that does not reduce but deny requests that exceed the available bandwidth.

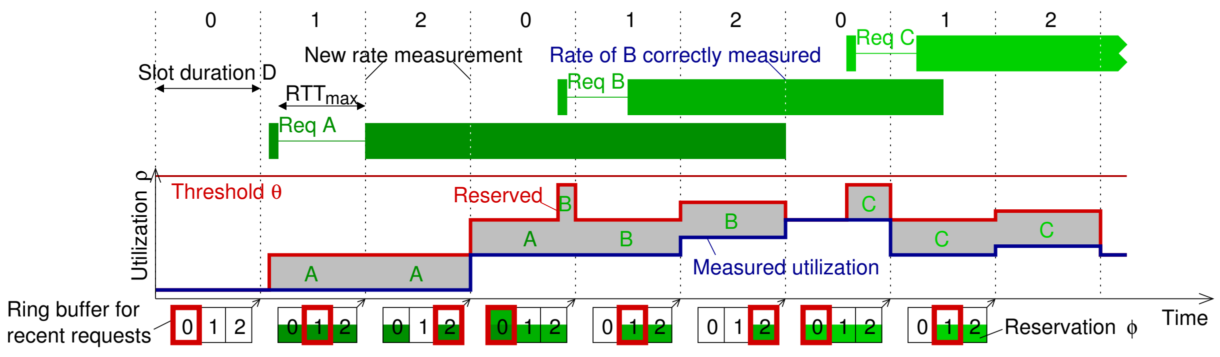

Fig. 7. Illustration of bandwidth measurement and recent requests history

This "without reduction" strategy would require less router processing, and it could also give incentives to hosts not to request for unnecessarily high data rates.

D. Handling Recent Approvals

Finally, routers must keep track of the approved Quick-Start requests in order to ensure that the output link will remain underutilized if the additional traffic of earlier Quick-Start requests arrives. A potential solution is to store temporal per-flow state, e. g., between the "Quick-Start request" and the "Quick-Start report" (see Fig. 1). As alternative, [2] proposes to keep track of the *aggregate* approved rate over recent rate measurement intervals. This does not require per-flow state and is thus more scalable.

The rate measurements and the history of recent approvals must be coordinated, as illustrated in Fig. 7: When a request is approved, the granted capacity must be stored until a new measurement is performed and until it is almost certain that this sample includes the data rate of the new flow, which may potentially occupy a significant share of the outgoing link capacity. This requires rather frequent rate measurements, i. e., a not too large value of D. In the example of flow B in Fig. 7, it takes three rate measurements after the request until the router can forget the Quick-Start request.

Assuming a worst-case round-trip time RTT_{max}, the minimum time to store a request is

$$H \geq RTT_{\text{max}} + D. \quad (7)$$

Choosing the parameter RTT_{max} involves a trade-off: If the time interval is too small, the router forgets recent rate approvals too quickly and thus may over-subscribe the available bandwidth. But if RTT_{max} is too large, the router is too conservative when approving Quick-Start requests.

We propose to use a ring buffer [24] for the storage of the recently granted aggregate bandwidth ϕ, as illustrated in Fig. 7: Granted data rates are added to all elements of the ring buffer. Each time a rate measurement is performed, the oldest element is emptied. Thus, a rate allocation is remembered during one complete cycle. From this follows the number of required buffer spaces as $\lceil \frac{RTT_{\text{max}}}{D} + 2 \rceil$.

VI. Performance of Quick-Start over Shared Links

A. Quick-Start Request Success Probability

On shared links, Quick-Start requests may or may not succeed, depending on the current load, other competing requests, and the admission control procedures. In the following, we analyze the "target algorithm" and approximate the success probability of Quick-Start requests as a function of the most important parameters. We start with the case that requests are not reduced. Then, we show how the analysis can be extended to the "with reduction" strategy.

One key challenge for the analysis is the online rate estimation: Both the measured traffic rate u and the corresponding link utilization $\rho = \frac{u}{r}$ typically vary on macroscopic and on microscopic scale. Even under a stationary offered load, subsequent rate measurements with a fine time granularity D can differ significantly, and the measured load ρ thus follows a distribution function $F_\rho(x)$.

The "target algorithm" will approve a request for rate q if the measured link utilization ρ, plus the normalized request rate and previous approvals, is not larger than threshold θ, up to which the link is considered as underutilized. As a consequence, the success probability p of a Quick-Start request for rate q is given by the probability that θ is not exceeded. Fig. 6 illustrates that there is the following relationship between the success probability p and the distribution function $F_\rho(x)$ of the measured link utilization:

$$p(q) = F_\rho\left(\theta - \tfrac{q}{r} - \phi(H)\right) \ . \quad (8)$$

$\phi(H)$ is the relative amount of capacity that has been allocated to Quick-Start in the last time interval $H = RTT_{\text{max}} + D$.

Volume-based rate measurements calculate u from the amount of data per time D. Thus, u corresponds to the number of stochastic arrivals within a given time. Even though the preconditions of the central limit theorem are not completely fulfilled here, one can assume that the measured utilization ρ is approximately normal distributed with mean μ and variance σ: $F_\rho = N(\mu, \sigma)$. In case of an underutilized link, the mean μ is identical to the offered load, while the variance σ depends on the arrival traffic pattern and the traffic metering.

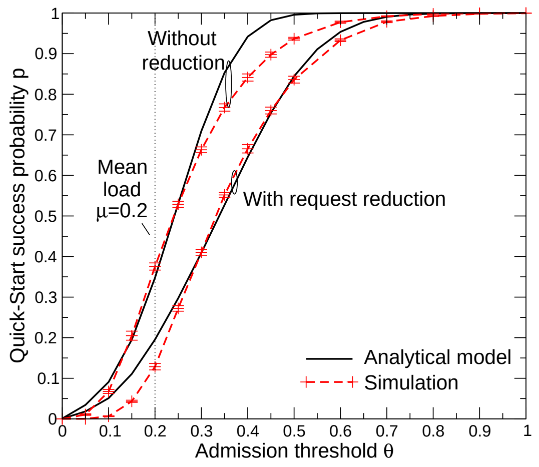

Fig. 8. Success probability for different thresholds ($RTT_{\text{max}} = 100\,\text{ms}$)

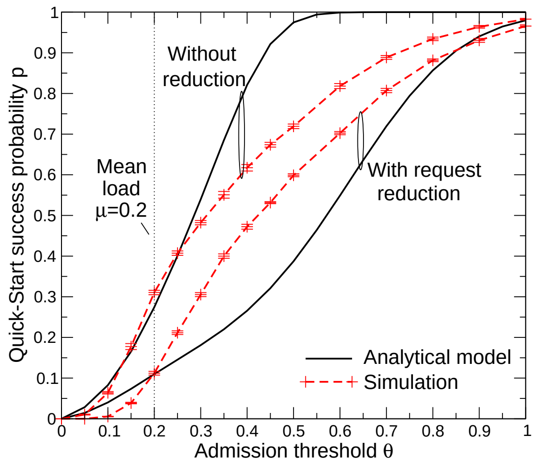

Fig. 9. Success probability for a conservative setting ($RTT_{\text{max}} = 500\,\text{ms}$)

If the arrival rate λ and the requested data rate q are known, the Quick-Start success probability can be determined by solving Eq. (8). However, the aggregated utilization $\phi(H)$ depends on the success probability of previous requests, too.

In the following, we use a simple heuristic to get an iterative solution: Let the random variable Z be the number of recent requests in time interval H, and suppose that ϕ is zero at the beginning of time interval H. Then, for $z = 0$ the success probability is $p_0(q) = F_\rho\left(\theta - \frac{q}{r}\right)$. If $z \geq 1$ requests of size q have been granted within H, the aggregated bandwidth is $\phi = \frac{q}{r} \cdot \sum_{i=0}^{z-1} p_i(q)$. The success probabilities $p_z(q)$ can thus be determined by recursion:

$$p_z(q) = F_\rho\left(\theta - \tfrac{q}{r} - \tfrac{q}{r}\sum_{i=0}^{z-1} p_i(q)\right) . \quad (9)$$

In case that Quick-Start requests arrive with negative-exponential inter-arrival time, Z is Poisson distributed: $f_Z(x) = P[Z = x] = \frac{(\lambda\,H)^x}{x!}\exp(-\lambda\,H)$. The mean Quick-Start success probability can then be derived as

$$p(q) = \sum_{i=0}^{\infty} f_Z(i)\cdot p_i(q) = \sum_{i=0}^{\infty} \tfrac{(\lambda\,H)^i}{i!}\exp(-\lambda\,H)\cdot p_i(q) . \quad (10)$$

For the case that request adaptation is enabled ("with reduction"), we consider in the following only a possible reduction by factor two. Similar to Eq. (8), the probability that half of the requested bandwidth is granted follows as:

$$p(\tfrac{q}{2}) = F_\rho\left(\theta - \tfrac{q}{2\,r} - \phi\left(H\right)\right) - p(q) . \quad (11)$$

Note that one could calculate the success probability of further reductions in a similar way. Now, the success probabilities $p(q)$ and $p(\frac{q}{2})$ can be derived by using two iterations for Eq. (8) and (11), similar to the procedure explained previously.

B. Comparison with Simulation Results

In order to study the impact of the parameters θ and H, we now consider the setup described in Section IV-A. In this setup, the mean load on the shared link is $\mu = \frac{S\cdot\lambda}{r_{\text{B}}} = 0.2$. This means that there is considerable bandwidth available for Quick-Start. For $D = 100\,\text{ms}$, the measured deviation of the load is $\sigma \approx 0.1$, i. e., $F_\rho \approx N(0.2, 0.1)$. We first use constant file transfers of size $m = 250\,\text{kbyte}$.

Figure 8 depicts the Quick-Start success probability $p = p(q)+p(\frac{q}{2})+\ldots$ as a function of the threshold θ for $RTT_{\text{max}} = 100\,\text{ms}$. If θ is less than the mean utilization $\mu = 0.2$, hardly any Quick-Start requests is successfully (approved rate > 0). But as soon as θ is slightly larger than the mean utilization, many Quick-Start request get approved. As to be expected, the success ratio is much larger if request reduction is allowed. Figure 8 reveals a close match of our analytical model and the simulation results.

However, our model is not precise in all cases: Fig. 9 shows result for the same setup, except that the minimum time to remember requests is now $RTT_{\text{max}} = 500\,\text{ms}$. Because of this conservative assumption about the maximum round-trip time, the granted capacity is reserved for a longer time H. This significantly reduces the Quick-Start success probability. The comparison of analysis and simulation shows that our model does not accurately predict the success probability if the admission threshold θ is slightly larger than μ. Under these constraints, a more complex modeling might be needed.

C. Impact on TCP Transfer Times

The transfer times without and with Quick-Start follow from Eq. (6). Without request reduction, the mean transfer time of a certain amount of data s is

$$T \approx p(q)\cdot T_{\text{QS}}\left(s, R_{\text{A}}, Q\right) + \left(1 - p\left(q\right)\right)\cdot T_{\text{SS}}\left(s, R_{\text{A}}\right) , \quad (12)$$

where $p(q)$ is the success probability and $R_{\text{A}} = \frac{L}{MTU}\cdot r_{\text{A}}$. If the router uses the "with reduction" strategy, one has to add further summands of the form $p(\frac{q}{2})\cdot T_{\text{QS}}\left(s, R_{\text{A}}, \frac{Q}{2}\right)$, etc. For random transfer sizes S with distribution $f_S(x)$, the mean transfer time could be determined from an integration over Eq. (12).

Figure 10 depicts T as a function of the threshold θ for $RTT_{\text{max}} = 100\,\text{ms}$. We only consider the case "with request reduction" here. Again, if θ is less than the mean utilization

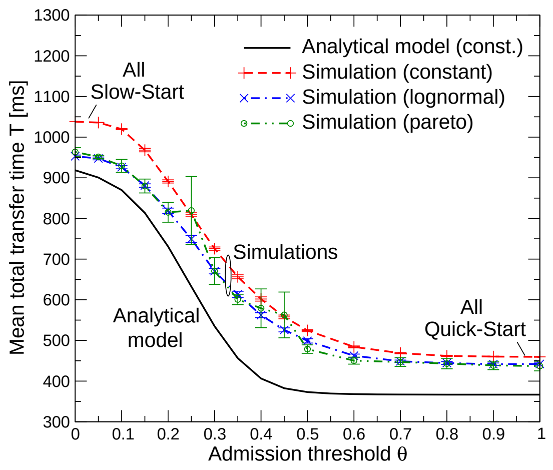

Fig. 10. Impact of the threshold on transfer times

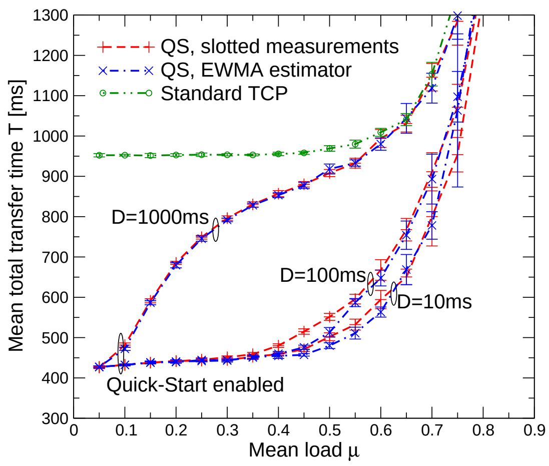

Fig. 11. Comparison of different rate measurement methods

0.2, the mean transfer time T is large, because hardly any Quick-Start requests gets approved. If θ is set to a value larger than the mean utilization, successful Quick-Start requests can significantly reduce the mean transfer time.

These results show that Quick-Start can result in a significant performance benefit on underutilized links even if only part of the link capacity are granted to Quick-Start. This outcome is quite insensitive to the transfer size distribution. The simulation results also match rather well to our analytical approximation, except for some constant offset. The reason for this offset is probably that the actual round-trip time over the shared link is larger than τ, which is not considered by the model in Section III.

D. *Impact of Different Rate Estimators*

We also study of the impact of different rate measurement techniques: Fig. 11 presents the results for the transfer time as a function of the offered load $\mu = \frac{S \cdot \lambda}{r_B}$, which is varied by changing λ. The threshold θ is here set to 0.95, and the transfer sizes are Lognormal distributed. As to be expected, for a rather high utilization it makes no significant difference whether to use Quick-Start or standard TCP, since only few Quick-Start request get approved. Note that $\theta = 0.95$ is a rather aggressive setting that admits Quick-Start requests even if the link is already highly loaded. For low or moderate utilization, there is again a considerable performance benefit. As to be expected, one can observe that the performance improves for more frequent rate measurements, i. e., small D, because the storage time of recent requests is smaller and the admission control is less conservative. However, more frequent measurements come along with the drawback that the measured load ρ may significantly fluctuate and thus require further postprocessing. The simulation results shown in Fig. 11 reveal that using an exponential-weighted moving average with weight factor 0.1 tends to slightly improve the overall performance, but the difference is not significant.

VII. Measurements under Linux

A. *Implementation Overview*

In [3], the Linux kernel has been extended to support the Quick-Start extension. This implementation within a real TCP/IP protocol stack allows to test Quick-Start in combination with real applications and in real network testbeds. The kernel patch completely implements Quick-Start for IP version 4 according to [1], [5], and it is currently based on Linux kernel version 2.6.20.11. The implemented features include the required host functions, i. e., processing of IP and TCP options, modification of TCP congestion and flow control, and the rate pacing mechanism. Also, the required router functions have been added to the Linux kernel IP forwarding, including metering of the current traffic over egress interfaces and admission control for Quick-Start requests. Fig. 12 gives an overview of the resulting modifications of the kernel code. More details about the Quick-Start implementation can be found in [3] and [25].

B. *Measurement Testbed Setup*

In the following, we report some initial measurements that have been obtained with the Linux Quick-Start implementation, in order to verify the analytical and simulation results presented in Section III. We consider a very similar setup, i. e., a certain amount of data is transferred after the connection setup with or without a Quick-Start request over an unloaded link. Different to Section III, now the connection originator receives data from the server. For this data transfer from the server to the client, no special mechansims are required to avoid interactions of Quick-Start and the TCP flow control [5].

The scenario is also illustrated in the message sequence chart in Fig. 13. Both client and server are realized as C programs and use straightforward socket calls. The server activates Quick-Start by setting a new socket option. The "server response time" between the completion of the three-way-handshake and the end of the data transfer can be easily measured within the client application program, without a need

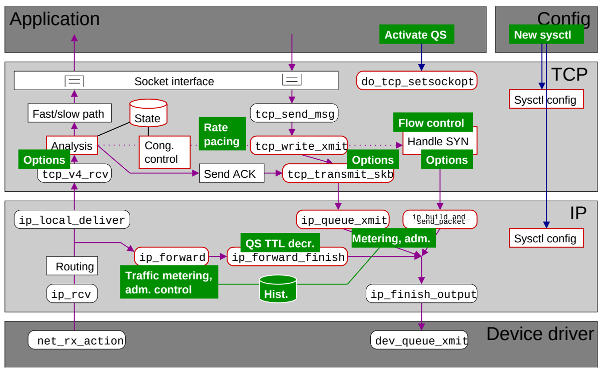

Fig. 12. Overview of Linux kernel Quick-Start implementation

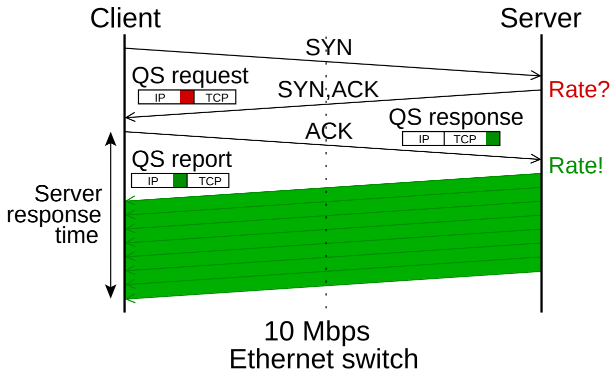

Fig. 13. Measurement scenario

for clock synchronization among the hosts. Each response time value is the average of ten consecutive measurements.

Client and server are Linux computers running the modified kernel. The kernel scheduler runs with $HZ = 1000\,\frac{1}{s}$, from which follows a rate pacing granularity of 1 ms. Client and server are interconnected by a $r = 10$ Mbps Ethernet switch, and each of them uses the "NetEm" [26] network emulation at the Ethernet interface to add a delay of 50 ms. As a consequence, the minimum RTT is again $\tau = 100$ ms. If Quick-Start is enabled, the server asks for an initial data rate of $q = 5.12$ Mbps, which is the maximum possible request rate in this setup. As the Linux kernel supports different TCP congestion control algorithms, we utilize both the "Reno" [27] and the "Cubic" [28] variant. As recommended in [5], the socket buffers have been increased to a large value (8 Mbyte) in order to avoid any limitation by the TCP flow control.

C. Quick-Start Measurement Results

Figure 14 shows the measured server response times both without and with Quick-Start. As to be expected, the result is very similar to the simulation results in Fig. 3. However, there are a couple of reasons why the absolute values slightly differ: First, the definition of the server response time T' is different, as one can see from Fig. 13. With the analytical model from Section III, T' can be expressed as follows:

$$T' = \frac{\tau}{2} + \Gamma \qquad (13)$$

Γ is either $\Gamma_{\mathrm{SS}}(s, R)$ or $\Gamma_{\mathrm{QS}}(s, R, Q)$. Second, Linux uses by default a so-called "Quick ACK" mechanism that acknowledges every segment during the beginning of Slow-Start [27]. This heuristic speeds up the Slow-Start, but it is not standard compliant. In the analytical model, $b = 1$ has to be used in order to take this into account. And finally, the MSS is slightly smaller ($L = 1452$ byte) because Linux uses by default the TCP timestamps option in every segment.

The comparison of Fig. 3 and Fig. 14 reveals that in the measurement the performance improvement of Quick-Start is smaller, mainly because of the "Quick ACK" mechanism. Nevertheless, mid-sized data transfers are accelerated at least by a factor of two. There is no significant impact of the congestion control algorithm, which is reasonable, since the behavior of "Reno" and "Cubic" is very similar after connection setup. The measurement results are slightly larger than the values given the analytical model of Eq. (13). This can be explained by small additional processing efforts that are not considered by the analytical model.

Finally, Figure 15 presents a trace of the data rates with Slow-Start or Quick-Start, respectively. The data rate values have been obtained by evaluating a packet trace and summing up the length of the corresponding IP packets within slots of duration $\Delta = 100$ ms. As one can see, with Slow-Start it takes at least one second to (almost) fully utilize the link capacity. Quick-Start is much faster and only requires about 300 ms to ramp up, part of which is caused by the temporary rate pacing with $q = 5.12$ Mbps.

Summing up, our initial measurement results clearly show that Quick-Start is a promising solution to improve transfer times, even though they have been obtained from a rather simple testbed setup. Further studies are needed to evaluate Quick-Start in a wider range of scenarios.

VIII. Conclusions and Future Work

Quick-Start is a TCP extension that allows hosts to cooperate with the routers along a path in order to determine a large initial sending rate. This paper studies the performance of Quick-Start compared to standard TCP Slow-Start by analytical models, by simulation, and by measurements in the Linux TCP/IP stack. We also consider different algorithms in routers and quantify the performance impact of key configuration parameters. Our results confirm that Quick-Start can significantly improve transfer times without requiring per-flow state in routers. Our analytical approximations could also be used to dimension the measurement-based admission control in the routers. Future work will include evaluations with more realistic application communication patterns and more complex network setups.

Acknowledgments

This work was partly funded by the German Research Foundation (DFG) through the Center of Excellence (SFB) 627. The author would like to thank Haiko Strotbek for his work on the Quick-Start implementation used in Section VII.

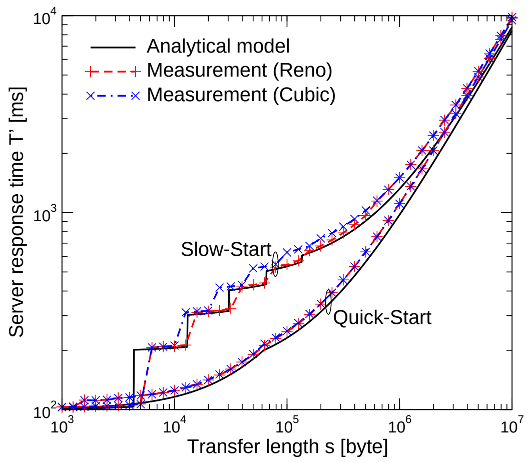

Fig. 14. Server response times ($\tau = 100\,\text{ms}$)

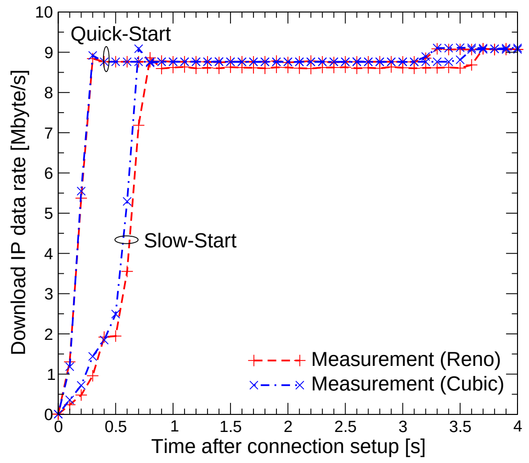

Fig. 15. Trace of data rates after connection setup

REFERENCES

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

[2] 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.

[3] H. Strotbek, "Design and implementation of a TCP extension in the Linux kernel," Diploma thesis (in German), University of Stuttgart, IKR, May 2007.

[4] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," IETF RFC 3390 (proposed standard), Oct. 2002.

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

[6] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo, "Using quick-start to improve TCP performance with vertical hand-off," in *Proc. 31st IEEE Conf. on Local Computer Networks*, Nov. 2006, pp. 897–904.

[7] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental evaluation of high-speed congestion control protocols," *to be published in IEEE/ACM Transactions on Networking*, 2007.

[8] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168 (proposed standard), Sept. 2001.

[9] D. Liu, M. Allman, S. Jin, and L. Wang, "Congestion control without a startup phase," in *Proc. 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet2007)*, Feb. 2007.

[10] B. Raghavan and A. C. Snoeren, "Decongestion control," in *ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-V)*, Nov. 2006.

[11] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, Aug. 2002.

[12] A. Falk, Y. Pryadkin, and D. Katabi, "Specification for the explicit control protocol (XCP)," IETF Internet Draft, work in progress, July 2007.

[13] A. Sathiaseelan and G. Fairhurst, "Use of quickstart for improving the performance of TFRC-SP over satellite networks," in *Proc. International Workshop on Satellite and Space Communications*, Sept. 2006.

[14] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proc. IEEE INFOCOM*, Mar. 2000.

[15] B. Sikdar, S. Kalyanaraman, and K. S. Vastola, "Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 959–971, 2003.

[16] A. Riedl, M. Perske, T. Bauschert, and A. Probst, "Investigation of the M/G/R processor sharing model for dimensioning of IP access networks with elastic traffic," in *First Polish-German Teletraffic Symposium*, Sept. 2000.

[17] S. Ben Fredj, T. Bonald, A. Proutiere, G. Régnié, and J. W. Roberts, "Statistical bandwidth sharing: A study of congestion at flow level," in *Proc. ACM SIGCOMM*, Aug. 2001.

[18] S. Bodamer, "Verfahren zur relativen Dienstgütedifferenzierung in IP-Netzknoten," PhD thesis (in German), University of Stuttgart, IKR, 2004.

[19] "IKR simulation library," http://www.ikr.uni-stuttgart.de/IKRSimLib/.

[20] A. B. Downey, "Lognormal and Pareto distributions in the Internet," *Computer Communications*, vol. 28, no. 7, pp. 790–801, 2005.

[21] L. Breslau, S. Jamin, and S. Shenker, "Comments on the performance of measurement-based admission control algorithms," in *Proc. IEEE INFOCOM*, Mar. 2000.

[22] R. Martin and M. Menth, "Improving the timeliness of rate measurements," in *12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, Sept. 2004.

[23] L. Burgstahler and M. Neubauer, "New modifications of the exponential moving average for bandwidth estimation," in *15th ITC Specialist Seminar*, July 2002.

[24] L.-O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 413–424, 2005.

[25] M. Scharf and H. Strotbek, "Experiences with implementing quick-start in the Linux kernel," Presentation at IETF 69, Chicago, IL, USA, July 2007.

[26] S. Hemminger, "Network emulation with NetEm," in *Proc. Australia's National Linux Conference (LCA)*, Apr. 2005.

[27] P. Sarolahti and A. Kuznetsov, "Congestion control in Linux TCP," in *Proc. USENIX Annual Technical Conference*, June 2002.

[28] I. Rhee and L. Xu, "Cubic: A new TCP-friendly high-speed TCP variant," in *3rd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet2005)*, Feb. 2005.