Evaluating Quick-Start for TCP

Pasi Sarolahti, Mark Allman, and Sally Floyd, February 18, 2005*

Abstract– This paper explores the Quick-Start mechanism, designed to allow transport protocols to explicitly request permission from the routers along the path to send at a higher rate than normally allowed by traditional congestion control mechanisms. If the routers are underutilized, they may approve the sender's request for a higher sending rate; otherwise the sender uses the default congestion control algorithms. This paper discusses some of the design issues of Quick-Start, and evaluates the potential benefits, costs and implications of Quick-Start in different networking environments. Using simulations, we evaluate several different algorithms that routers could use to process a Quick-Start Request. This evaluation explores tradeoffs between the fraction of Quick-Start requests that are approved and the fraction of approved Quick-Start requests that result in increasing network congestion. In addition, the paper discusses the security implications of using Quick-Start and some possible mitigations for the vulnerabilities.

1 Introduction

A fundamental aspect of communication in general-purpose, best-effort packet-switched networks is determining an appropriate *sending rate*. The appropriate sending rate depends on the characteristics of the network path between the two peers (bandwidth, propagation delay, etc.), as well as the amount of load being placed on the network by others at the given time. Traditionally, TCP has used a set of congestion control algorithms for determining this rate [10]. The problem we tackle in this paper is how a particular connection that is under-utilizing (or, even not using) a network path can rapidly increase its transmission rate to take advantage of the available capacity more rapidly than allowed by TCP's traditional congestion control algorithms.

The first place the issue of determining an appropriate sending rate occurs is when choosing an *initial* rate. The current method for choosing an initial sending rate in the Internet is to use TCP's [19] slow start algorithm [10, 3]. TCP controls the sending rate using a congestion window (*cwnd*), which bounds the amount of data that can be transmitted into the network before receiving an ACK. Slow start initializes *cwnd* to a small value (1–4 segments, per [3, 2]) that is assumed to be an appropriate starting point for the vast majority of situations. In each subsequent congestion-free round-trip time (RTT), *cwnd* is increased exponentially by 50–100% (depending on whether the receiver uses delayed acknowledgments [8, 3] and whether the sender uses packet or byte counting [4] to increase *cwnd*). Slow start is terminated when either (i) the sender exhausts the data to be transmitted, (ii) congestion is detected or (iii) *cwnd* reaches the receiver's advertised window. After slow start has probed for an appropriate operating point, additive-increase, multiplicative-decrease (AIMD) congestion control governs the remaining transmission [10, 3].

In many environments, slow-start can require a significant number of RTTs and require a large amount of data to open *cwnd* sufficiently to fully use the available bandwidth. For example, even in the best case with byte-counting and an initial window of four packets, slow-start takes $log_2 N - 2$ round-trip times and requires sending $N - 3$ packets before reaching a congestion window of N packets.

Probing for the available bandwidth makes sense when there is general contention for the resources at a congested point in the network path. However, when a network path is uncongested and largely under-utilized, this slow probing process introduces unnecessary delay for the application. For a connection over an under-utilized path, there might be enough bandwidth for the connection to complete its entire data transfer in one round-trip time. In this paper we examine Quick-Start, a proposed mechanism for end nodes to request permission from routers along the path to use a higher sending rate [11]. Quick-Start has the potential to alleviate the delay of slow-start for connections in under-utilized environments.

Although Quick-Start could be used with a number of transport protocols, in this paper we mainly consider its use with TCP. Quick-Start is described in detail in Section 3, but the process is generally that a TCP connection sends a packet that includes a Quick-Start Request in an IP option containing the requested sending rate, say X bytes/sec. Each router along the path either indicates agreement with the request, lowers the requested sending rate or implicitly signals that the Quick-Start option was not processed (and, hence, the request will not be approved). The data receiver reports the information received in the Quick-Start Request back to the sender using a Quick-Start Response in a TCP option, and the data sender determines if all of the routers along the path have agreed to the request and sets the sending rate appropriately.

The assumption behind Quick-Start is that routers will

*This material is based in part upon work supported by the National Science Foundation under Grant No. 0205519. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

only approve Quick-Start requests when they are under-utilized. Thus, Quick-Start should be generally safe to deploy in general purpose networks, with a negligible risk of causing network congestion. However, because Quick-Start requires support from all routers along the path, this could present a high bar to deployment in the general Internet. Possible initial deployments could come in (i) those Intranets and operator networks with large amounts of under-utilized bandwidth and (ii) cellular wireless networks (such as GPRS/EDGE [21]) with bandwidth of up to 384 Kbps, but with long round-trip delays. Based on the investigation presented in this paper, Quick-Start is expected to be of benefit in both these cases.

As noted above, Quick-Start is, broadly speaking, useful any time a connection is significantly under-utilizing the network path and has the data required to considerably increase the transmission rate. The path from broad notion to mechanism is not clear-cut and future work in this area is required. However, there are a few concrete cases where the connection is likely to be significantly under-utilizing the capacity and could benefit from Quick-Start. As discussed above, at the beginning of a TCP connection when little if any knowledge about the network path exists the end hosts may be able to use Quick-Start to transmit at a higher initial rate. Similarly, after an explicit message informs a TCP connection of a change in network attachment point (e.g., due to the use of Mobile IP), the connection again has little information about the (new) network path and might be able to use Quick-Start to use a higher sending rate than would otherwise be appropriate. In addition, after "idle" periods in a connection, TCP's understanding of the available bandwidth is stale and Quick-Start may be helpful in re-establishing a higher sending rate when data transmission begins again. This last case starts down the path of trying to assess when a connection is in fact "under-utilizing" the network path. Questions that pop to mind are: What does "idle" mean? Silent or mostly silent with low-rate control messages? How long does the "idle" period have to be before Quick-Start is again appropriate? Is it appropriate to use Quick-Start when a connection is not idle at all but just transmitting at a low rate due to an application limitation, followed by a spike in the amount of data to be transmitted? In this paper we concentrate on the first order question of Quick Start's efficacy in the clear-cut case of determining an initial sending rate, leaving the thornier questions of precisely when Quick-Start should be employed to future work.

While Quick-Start is a component of congestion control, Quick-Start is not a complete congestion control mechanism, and it is not intended as a replacement for TCP's standard congestion control. Quick-Start is also not a Quality of Service (QoS) or resource reservation mechanism. Quick-Start is in fact most effective in those under-utilized environments where congestion control is not the overriding issue, and where QoS mechanisms are needed the least. In the subsequent sections we show this via simulation.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 details the Quick-Start mechanism and discusses design issues. Section 4 discusses the potential costs and benefits of using Quick-Start. Section 5 describes the simulation setup used our study. Section 6 illustrates the potential advantages and disadvantages of Quick-Start and shows its performance in specific situations. Section 7 discusses the handling of Quick-Start Requests in the routers and evaluates several algorithms that could be employed by routers. Section 8 outlines the possible vulnerabilities of Quick-Start to denial-of-service attacks and potential coping techniques. Finally, Section 9 offers conclusions and future work.

2 Related Work

There have been a number of proposals for faster variants of TCP slow-start that do not use explicit feedback from routers. For example, SwiftStart [17] would use the first volley of packets sent during slow start to estimate the bottleneck bandwidth, and then use that estimate as the basis for a rapid increase in the congestion window.

There are also proposals for sharing information about network conditions between connections, ranging from TCP Fast Start [16] to the Congestion Manager [5], that would allow a new connections to start with a larger *cwnd*, based on the assessment of the network path conducted by previous connections.

Additional proposals call for other new mechanisms for explicit congestion-related feedback from routers to end-nodes. In Explicit Congestion Notifications (ECN) [20], the only current mechanism in the IP protocol for explicit feedback from routers to end-nodes, routers use the ECN field in the IP header to indicate congestion explicitly, instead of relying on packet drops. In contrast, the Anti-ECN proposal [14] would allow the sender to increase as fast as slow-start over an uncongested path, even in the middle of a transfer, with routers setting a bit in the packet header to indicate an under-utilized link.

XCP (Explicit Control Protocol) [13] is a proposal for a new congestion control mechanism based on explicit and fine-grained per-packet feedback from the routers over the course of the entire transfer. XCP is similar to Quick-Start in that the routers are explicitly involved in feedback on the senders' allowed transmission rates, but the goals of the two schemes are different. While XCP provides a full-fledged congestion control mechanism, Quick-Start, in some sense, provides just the opposite. Quick-Start provides for a brief check to determine whether the path is underutilized, allowing the sender to start or move to a high sending rate.

3 Quick-Start

Quick-Start is a collaborative effort between end hosts and routers. This section describes the details of Quick-Start,

and discusses the Quick-Start requirements. [11] gives a detailed specification of Quick-Start.

3.1 Quick-Start Processing at the Sender

The Quick-Start *Rate Request* is initialized by the sender to the desired sending rate in bytes per second (Bps). The sender also initializes a *Quick-Start TTL* to a random value and saves the difference between the initial Quick-Start TTL and the initial IP TTL as *TTLDiff*. As discussed in the next subsection, the routers along the network path between the sender and receiver alter the Request, as appropriate. When the Quick-Start Request arrives at the transport receiver, the receiver echoes the rate request back to the sender along with the difference between the Quick-Start TTL and the IP TTL, *TTLDiff′*, in an option in the transport header. Upon reception of an echoed Quick-Start Rate Request the sender verifies that all routers along the path have approved the Quick-Start Request by comparing *TTLDiff* and *TTLDiff′*. If these two values are not the same then the request was not approved by all routers in the network path and data transmission will continue using TCP's standard algorithms.

When *TTLDiff* and *TTLDiff′* match, the TCP sender then calculates the appropriate *cwnd* based on the approved sending rate and measured round-trip time as follows:

$$cwnd = \frac{Rate * RTT}{MSS + H}, \quad (1)$$

where *Rate* is the approved rate request in Bps, *RTT* is the recently measured round-trip time in seconds, *MSS* is the maximum segment size for the TCP connection and *H* is the estimated header overhead for the connection in bytes. The TCP sender paces out the Quick-Start packets at the approved sending rate over the next RTT[1]. Upon receipt of an acknowledgment for the first Quick-Start packet, the TCP sender returns to ACK-paced transmission.

3.1.1 Knowing the Rate to Request

One of the problems of Quick-Start is that unnecessary or unnecessarily-large Quick-Start Requests can "waste" potential Quick-Start bandwidth; because routers must keep track of the aggregate bandwidth represented by recently-approved Quick-Start requests (so that the router does not over-subscribe the available capacity), each approved request reduces the chances of approval for subsequent requests. Ideally, a sender should not use Quick-Start for data streams that are not expected to benefit from it, such as those that have only a few packets of data to send. The TCP sender should, in theory, also avoid requesting an unnecessarily high sending rate. However, it can be difficult for the TCP sender to determine how much data will ultimately be transmitted and therefore to form a reasonable rate request. For example, in request-response protocols such as HTTP [6], the server does not know the size of the requested object during the TCP handshake; it hasn't yet received the data request. Once the web server does know the requested object, the application would need to determine the size of the object and then inform TCP as to how many bytes will be sent, because the objects are rarely written to the TCP socket buffers in a single atomic call. Even if the web server went to all of this trouble with persistent HTTP connections there may still be more data that the web server does not yet know about. Finally, sometimes the application cannot even obtain the size of an object because the object is being read from a pipe or some live source. In Section 6.2 we illustrate the problems of not making a reasonably accurate rate request and offer some strategies for coping.

[1]Note that TCPs are required to implement an additional timer for paced transmission when using Quick-Start.

3.2 Quick-Start Processing at Routers

A router that receives a packet with a Quick-Start Rate Request has several options. Routers that do not understand the Quick-Start Request option simply leave the option untouched, ultimately causing the Quick-Start Request to be rejected because *TTLDiff′* will not match *TTLDiff*. Routers that do not approve the request can either leave the Quick-Start Request option untouched, zero the Rate Request, or delete the option from the IP header. Routers that approve the rate in the request decrement the Quick-Start TTL and forward the packet. Finally, a router can approve a rate that is less than the rate in the request by reducing the rate, as well as decrementing the Quick-Start TTL.

Routers should only approve a Quick-Start Request when the output link has been underutilized over some recent time period. In order to approve a Quick-Start rate request, a router generally should know the bandwidth of the outgoing link and the utilization of the link over a recent period of time. At a minimum, the router also must keep track of the aggregate bandwidth recently approved for Quick-Start Requests, to avoid approving too many requests when many Quick-Start Requests arrive within a small window of time. Section 7 discusses in more detail the range of algorithms that could be used by routers in approving or denying a Quick-Start request.

Finally, we note that in this paper we discuss router algorithms in terms of "allocating" capacity, but that our notion of an "allocation" is quite informal. Quick-Start routers do not in fact reserve capacity for a particular flow and then police the usage to ensure that the given flow is able to use the granted capacity. Rather, the router simply tracks the aggregate amount of promised capacity (in the recent past) in an effort not to promise more than the output link can absorb. If, however, a burst of unexpected traffic arrives the Quick-Start "allocations" may prove to be empty promises when the end hosts attempt to use the granted bandwidth and detect congestion.

4 Costs, Benefits and Implications

This section discusses some of the potential costs, benefits and implications of adding Quick-Start to a network.

Increased Periods of Congestion: The general notion of Quick-Start is that it should be approved only in situations where the network path is under-utilized, thus allowing a connection to quickly use spare capacity. Therefore, the correct use of Quick-Start should not result in increased packet drop rates in the network. In other words, Quick-Start should not *cause* congestion, but rather should allow a connection to quickly use the *spare* capacity in the path. In Section 6 we show that proper use of Quick-Start does not increase the aggregate drop rate in a network. The flip-side is that bugs in the Quick-Start process could introduce inappropriate traffic to congested situations. To mitigate this, the drop of a Quick-Start packet causes the TCP sender to make a full reset to standard slow start.

Misbehaving Nodes and Routers: Quick-Start may provide new ways for two types of misbehavior. First, misbehaving receivers or routers could try to "game" Quick-Start to benefit the connections using Quick-Start. Non-conformant routers or hosts might try to modify the Quick-Start messages to benefit particular connections. For instance, a receiver may increase the rate given in an arriving Quick-Start Request before echoing it back to the sender in an effort to increase the connection's performance. Similarly, a router close to the sender and acting on the sender's behalf (a "performance booster") could increase the approved sending rate and/or adjust the reported *TTLDiff′* from the receiver to match the original *TTLDiff* in an effort to mask the network's lack of Quick-Start savvy. While it is possible to attempt to game Quick-Start, it is not without risk of lower performance, since TCP reverts to standard slow start if overzealousness results in packet drops in the Quick-Start window — effectively slowing the data transmission (as illustrated in Section 6). A second type of misbehavior comes from attackers attempting to prevent legitimate use of Quick-Start. This aspect of Quick-Start is further discussed in Section 8.

Added complexity at routers and end-nodes: One of the main costs of Quick-Start is that the required changes to both end-hosts and routers may moderately increase implementation complexity. For end-hosts the additional complexity may be justified by (i) the possible benefits of Quick-Start and (ii) that end hosts often have spare processing capability (although, this is not universally true — especially for busy servers). However, the additional complexity at routers can be a difficult issue, since performance and scalability requirements in routers have to be carefully balanced. Packets containing a Quick-Start Request represent an extra burden for routers and could result in extra delay for end-hosts. Of course, all packets would not contain Quick-Start Requests. Additionally, Quick-Start should only be approved in times of under-utilization and therefore the routers may be able to perform an efficient quick check of the utilization and only act on Quick-Start requests when the router is under-utilized (and, can likely better absorb the additional processing requirement). The practical implications for Quick-Start on real routers requires solid assessment, but is beyond the scope of our initial study.

Interactions with Middleboxes: It is known that there are middleboxes in the current network that drop packets containing known or unknown IP options [15]. This could result in significant delay for connections using Quick-Start requests, as packets using Quick-Start requests would have to be retransmitted without the Quick-Start Request Option (and if the option is transmitted on a SYN segment the initial retransmission timeout of 3 seconds [18] makes this a lengthy process). One consequence is that initial deployments of Quick-Start may be in controlled environments, where it is known that packets with Quick-Start options would be forwarded.

Deployment: An additional downside of the Quick-Start approach is that the scheme is not conducive to incremental deployment. Since both end systems and all the routers along some path have to support Quick-Start for the mechanism to work there is quite a high barrier to general use. We expect that initial deployments of Quick-Start would happen within closed networks whereby hosts and routers both have an interest in aiding performance.

5 Simulation Setup

In the following sections we use the ns-2 simulator to explore various facets of Quick-Start. We use a network comprised of three routers, R_1–R_3, arranged in a chain. The two links between the routers have bandwidth of L_{bw} and a one-way link delay of L_d. Unless otherwise noted, L_{bw}=10 Mbps and L_d=20 msec. The routers employ drop-tail queuing with a maximum queue size of 150 packets.

For most simulations, web clients and servers are connected to the ends of the network (to R_1 and R_3) with dedicated 1000 Mbps links with a mean one-way link delay of 12 msec and a maximum delay of 110 msec. The actual link delays are chosen to give a range of round-trip times that roughly matches those from measurements, using the process from [9]. A varying number of web servers, N, are connected to R_1 with a corresponding number of web clients connected to R_3. The measurements presented in the subsequent sections all refer to the traffic from the web servers connected to R_1. We also attach $\frac{N}{2}$ web clients to R_1 and $\frac{N}{2}$ web servers to R_3 to provide background traffic on the return path. When Quick-Start is enabled, all web servers attempt to use Quick-Start. The standard web traffic generator included with ns-2 is used in our simulations, with the following parameter settings: an average of 30 web pages per session, an inter-page parameter of 0.8, an average page size of 10 objects, an average object size of 400 packets and a ParetoII shape parameter of 1.002. We

use HTTP/1.0-like transactions, with one web object per TCP connection. These parameters, particularly the average object size, are not picked to match realistic traffic distributions, but rather to explore Quick-Start's impact on a wide swatch of connection sizes. Our web traffic simulations are run for 150 seconds.

A few simulations make use of a single transfer at a time. These simulations use FTP to transfer a file of a given size over the network given above with no reverse traffic present.

Finally, all TCP connections use ns-2's *sack1* TCP variant with an initial *cwnd* of 3 segments (per [2]), an MSS of 1460 bytes, an advertised window of 10,000 segments[2], and the receiver acknowledging each segment.

All simulations presented in the remainder of the paper use this setup unless otherwise noted. Simulation scripts will be on-line on the Quick-Start web page [1].

6 Connection Performance

In this section we explore when Quick-Start is and is not of benefit. In addition, we consider how to choose the Quick-Start request size, the implications of Quick-Start on aggregate network traffic and the implications of Quick-Start failures.

6.1 Ideal Behavior

In an ideal Quick-Start scenario over an under-utilized network path, the TCP sender would be able to transmit as much of its data in the initial congestion window as the spare network capacity can absorb. Figure 1 illustrates the ideal Quick-Start behavior by displaying time-sequence plots of two connections[3]. The first connection is a standard TCP connection that uses slow start to begin transmission (with an initial *cwnd* of 3 segments, per [2]). The second connection on the plot shows a case where an approved Quick-Start Request allows the sender to transmit 25 of its 30-packet transfer in the first round-trip time. When the first acknowledgment for data arrives at the TCP sender, the sender continues in slow-start, sending two packets for each acknowledgment. The connection using Quick-Start completes in just over half the time required by the non-Quick-Start connection.

Equation (2) gives the number of round-trip times, *NumRtts*, required for transmitting *N* packets of data in TCP slow-start assuming an ACK for each segment transmitted[4], in addition to the initial SYN exchange, given an initial congestion window of *W* packets (and where N and W are both at least one segment).

[2]This is high enough to make the advertised window a non-issue in our simulations.

[3]In this scenario the link bandwidth was 384 Kbps and the round-trip delay one second, roughly motivated by a GPRS/EDGE wireless scenario [21].

[4]This assumes that there is no congestion in either direction and the receiver's advertised window does not constrain the congestion window.

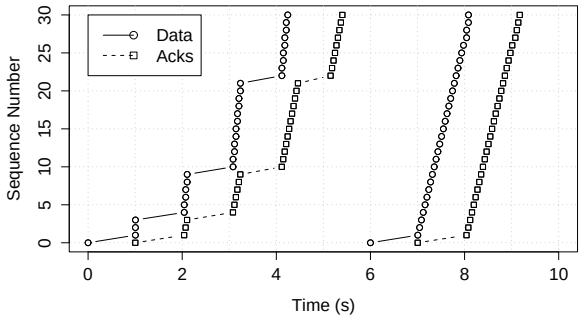

Figure 1: Normal TCP Slow-Start (left) vs. Quick-Start (right).

$$NumRtts = \left\lceil log_2 \left(\frac{N}{W} + 1 \right) \right\rceil \quad (2)$$

From this equation we note the clear attraction to maximizing W as much as is appropriate over a given network path.

Next we use the ns-2 simulator to investigate the ideal impact of Quick-Start. We use a simple scenario with capacity set at either 384 Kbps or 100 Mbps, various link delays, routers with unlimited buffers, routers willing to allocate 90% of their capacity to Quick-Start requests and TCP making Quick-Start Requests of 20 MB/sec. Figure 2 shows the results of the simulations. Although the simulation scenario is not necessarily realistic, it illustrates the potential impact of using Quick-Start. The results confirm the theoretical analysis above, showing that increasing the initial *cwnd* aids performance — especially for medium-sized transfers that are close to the delay-bandwidth product of the network path. In addition, the plots show that Quick-Start is less beneficial for excessively short or long transfers. Short transfers leave little room for improvement since they take little time. The performance of the long transfers in these simulations is dictated by the bottleneck link rate. Therefore, the longer the connection lasts the less impact the startup scheme has on overall performance since the connections perform identically after the startup phase. Our results are similar to earlier results from Sundarrajan [22].

6.2 The Size of the Quick-Start Request

We next consider how the sender chooses the Quick-Start request size, and how the size of Quick-Start requests affects the aggregate usefulness of Quick-Start. As discussed in Section 3.1.1, an ideal Quick-Start request would contain the precise sending rate the connection would like to use. However, knowing such a sending rate is non-trivial and depends on a number of factors. A simple Quick-Start implementation for TCP could send a fixed Quick-Start request each time a request is transmitted. This would not be unreasonable for initial Quick-Start requests, since in many cases the TCP sender has no knowledge about the application or the network path when the TCP SYN segment is

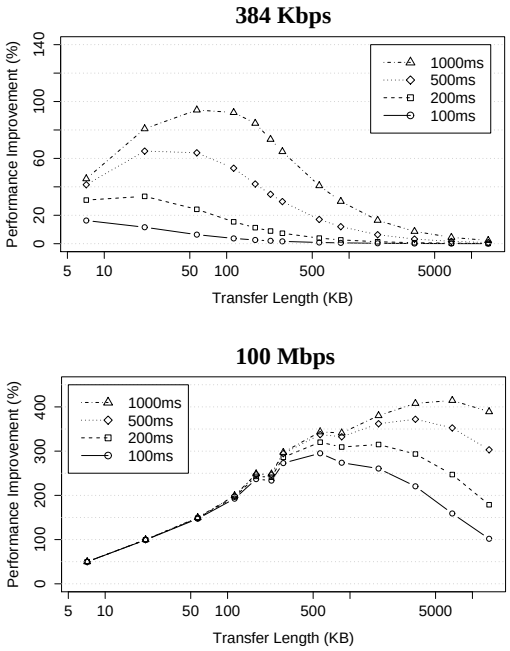

Figure 2: Relative improvement with Quick-Start, for a 384 Kbps link and a 100 Mbps link with a range of propagation delays.

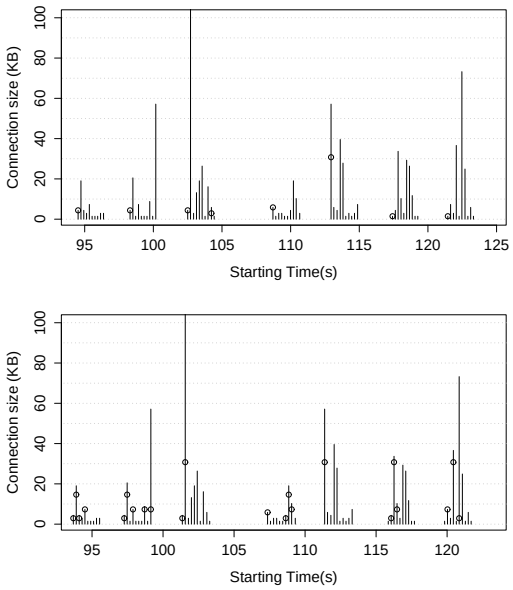

Figure 3: TCP connection lengths and starting times. Connections with Quick-Start packets are marked with a circle.

sent. For Quick-Start requests sent in the middle of a connection, e.g., after an idle period, the sender may be able to make a more informed Quick-Start Request.

To illustrate the problem with overly large Quick-Start requests we simulate two scenarios involving web traffic that uses one TCP connection for each web object transferred. Figure 3 shows the results. Each vertical line on the plots represents a separate TCP connection's length, and each circle indicates the quantity of Quick-Start data transmitted over the given connection. In the first case (top plot), TCP connections use a static Quick-Start request of 2 MB/sec for each connection. In the second scenario (bottom plot) the requests are ideal (even if unrealistic) for the amount of data the given connection will ultimately transmit. In addition, Quick-Start is not used if the connection is able to send all data in 3 segments (per the initial *cwnd* allowed by [2]). This example uses an average web object size of 60 packets.

As shown in the top plot, in this scenario Quick-Start requests are generally granted for only the first connection in each group. The router is generally unable to approve requests of later connections in each group, because the first connection is granted all of the available Quick-Start bandwidth even though the first connection cannot use such a large allocation. As a result, the extra allocation is "wasted", in that subsequent Quick-Start requests are denied unnecessarily. The bottom plot shows that when making ideal Quick-Start requests the Quick-Start requests are approved more often because there are fewer wasted approvals.

While the ideal case above is preferable, TCP connections do not, in general, have enough information to make ideal requests. However, there are several ways systems can cope. First, if an end-host is configured to understand the maximum capacity of its last-mile hop, C bytes/sec, requests could be chosen to be no larger than C. Going even further, a policy decision could be made to disallow any one TCP connection from using more than some fraction of the capacity and that could be used as an upper bound on the Quick-Start request (e.g., on a large web server). In addition, a sender could leverage the size of the local socket buffer, S bytes, and the receiver's advertised window, W bytes, when choosing a request size[5]. Given an RTT of R sec[6] TCP can send no faster than $min\ (S, W)\ /\ R$ bytes/sec (assuming W is non-zero and using S if it is). Finally, and more speculatively, if an application informed the sender of the size of a particular object (when known), say O bytes, the sender could request precisely the rate required to transmit the object in a single RTT as $(O + (O/MSS) * H)/R$ bytes/sec for a given MSS size and estimated header size of H bytes. While these techniques do not necessarily provide for an ideal Quick-Start request they could well provide a more reasonable request than simple picking a static rate for all cases.

6.3 Loss of Quick-Start Packets

We now consider the response of a TCP sender to the loss of a Quick-Start packet, that is, a packet sent in the RTT after

[5]When sending a request in the initial SYN segment of a connection the sender will not know the peer's advertised window.

[6]Or, an approximation if the connection has not yet taken an RTT measurement.

a Quick-Start Response triggers an increased sending rate.

Routers should only approve a Quick-Start Request when the output link is significantly underutilized and therefore there should be few congestion losses due to transmitting at the rate prescribed by Quick-Start. However, it is possible for there to be losses of Quick-Start packets because the allocations are not reservations. If a Quick-Start packet is lost after an approved Quick-Start Request, we call this a *Quick-Start failure*. This situation can arise for a number of reasons, for instance because a burst of traffic arrives at a router immediately after the router approves a Quick-Start Request, or because a buggy or broken router simply approves all Quick-Start requests or mis-calculates the rate that should be approved.

Generally, after detecting a lost packet, the TCP sender halves its congestion window and transmission continues using the congestion avoidance algorithm [10, 3], increasing the congestion window by roughly one segment each round-trip time. However, when a Quick-Start failure occurs, the sender cannot make strong assumptions about the current path capacity; in particular, the sender cannot fall back on the fact that a congestion window of half the current size was successfully transmitted in the previous round-trip time, as is the case during slow-start. As a result, halving the congestion window would not necessarily be an appropriate response to a Quick-Start failure. Instead, as specified in [11], after a Quick-Start failure the TCP sender returns to slow-start, using the default initial window, as it would have done if Quick-Start had not been approved.

Figure 4 shows time-sequence plots of several different TCP variants to illustrate TCP's response to a loss of a Quick-Start packet. The top plot in the figure shows a Quick-Start failure followed by fast retransmit and fast recovery (i.e., a simple halving of the congestion window). The second figure shows a Quick-Start failure followed by the proposed response of a slow start from the standard initial congestion window. Finally, the bottom plot shows a connection using standard slow start without Quick-Start. Because after fast recovery the congestion window increases in a linear fashion while Slow-Start increases *cwnd* exponentially, the Slow-Start response may find the appropriate sending rate faster than congestion avoidance, and hence offer better performance (as is illustrated in the figure). In addition, depending on the size of the congestion window used by Quick-Start, a simple halving may not be enough to alleviate congestion within the network and so several multiplicative decreases could be required before TCP finds an appropriate value for *cwnd*. With a Slow-Start response to a Quick-Start failure, the sender loses roughly two round-trip times because of the Quick-Start failure,[7] compared to a transfer without Quick-Start (shown in the bottom graph of Figure 4). While a Quick-Start failure should be a rare event, Figure 4 shows that standard slow start without Quick-Start can be a better choice over a path with a badly behaving or buggy router.

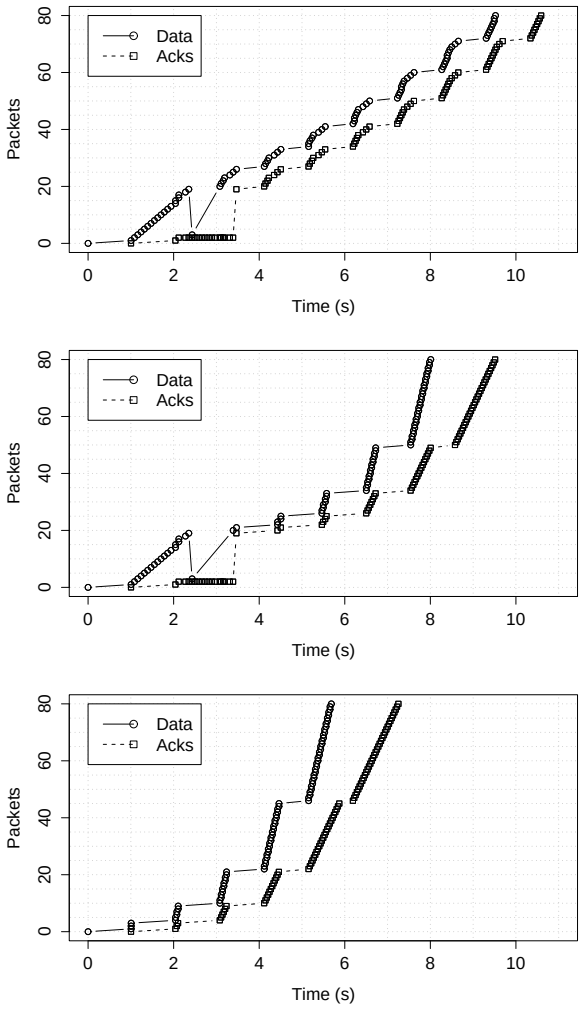

Figure 4: The TCP Response to a Quick-Start Failure.
- Top: Halving the window after a loss.
- Middle: Slow-Start after a loss.
- Bottom: Slow-Start without Quick-Start, without losses.

Finally, we note that ECN [20] can be used with Quick-Start. As is always the case with ECN, the sender's congestion control response to an ECN-marked Quick-Start packet is the same as the response to a dropped Quick-Start packet, thus reverting to slow start in the case of Quick-Start packets marked as experiencing congestion.

6.4 Aggregate Impact of Quick-Start

Because Quick-Start requests are only approved when the output link is significantly underutilized, Quick-Start should have little effect on the long-term aggregate utilization and drop rates on a link. In particular, when link utilization is high, routers should not approve Quick-Start re-

[7] This assumes SACK-based loss recovery that can detect and repair multiple losses within one RTT [7]. More generally, the connection is lengthened by one Quick-Start RTT and the time required by the loss recovery operation when compared to standard TCP.

quests; thus, Quick-Start is not a mechanism designed to help a router maintain a high-throughput low-delay state on the output link. In Section 7 we study various methods for routers to use to choose whether to approve Quick-Start requests and how much capacity to grant each request. In addition, we illustrate the implications of using Quick-Start when the router is not significantly under-utilized.

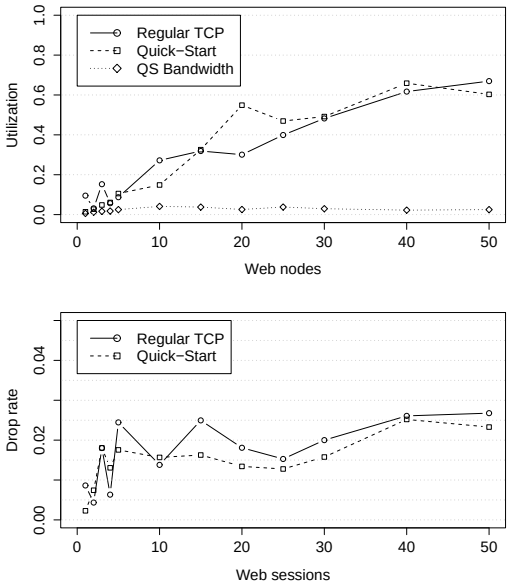

Figure 5: Comparison of utilization and drop rates with and without Quick-Start, with a 10 Mbps shared link.

For the traffic models used in this paper, the amount of data requested by a user is independent of whether Quick-Start is used, and independent of the fate of the Quick-Start requests. While the use of Quick-Start or particular allocations from the routers will have an impact on the time required for particular transfers, the aggregate amount of data requested is not effected. Given this model, although the use of Quick-Start might be of great benefit to the individual user, Quick-Start should have little effect on the long-term aggregate link utilization or packet drop rates.

However, it is possible that the successful use of Quick-Start would increase the amount of data sent and received by each user. For example, some users could have a fixed amount of time available for web browsing, rather that a fixed amount of data to send and receive. In this case, the use of Quick-Start could result in an increase in aggregate utilization in under-utilized scenarios. Even in this case, however, the use of Quick-Start should not affect the utilization and loss rates over paths that are not under-utilized, because in these scenarios Quick-Start requests should not be approved by the routers.

Figure 5 shows the overall utilization and aggregate drop rates with and without Quick-Start as a function of the number of web sessions, for a simulation scenario with web traffic with an average object size of 400 packets (as described in Section 5) on a 10 Mbps shared link. As shown in the

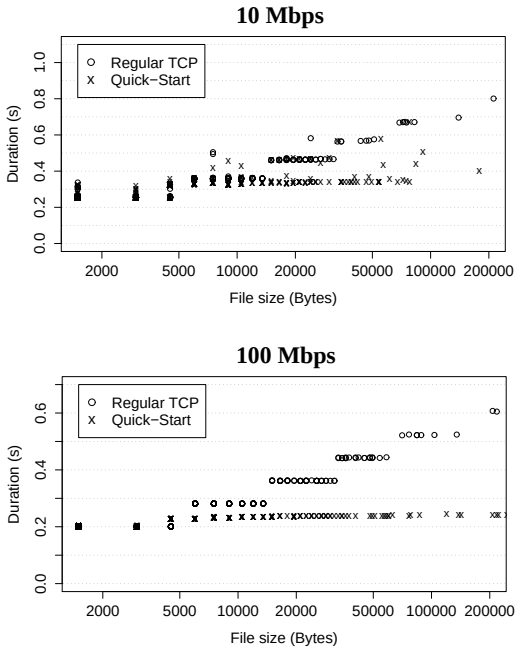

Figure 6: Per-connection performance with and without Quick-Start, with 10 Mbps and 100 Mbps shared links and three web sessions.

figure, the utilization and drop rates are largely independent of whether or not Quick-Start is employed. The line labeled "QS Bandwidth" in the top graph of Figure 5 shows the bandwidth used by Quick-Start packets in the simulations using Quick-Start — indicating that Quick-Start is being put to use at the beginning of transmission. We also conducted simulations with a smaller average web object size (of 60 packets) and obtained similar results.

Figure 6 shows per-connection performance of all traffic involved in a simulation of 3 web servers. Each point on the plot represents the duration of a single connection, with the point type indicating whether Quick-Start is used. The top plot shows the results from a simulation run over a 10 Mbps link while the bottom plot uses a 100 Mbps link. For medium to large transfers the plots show Quick-Start improves performance — by a factor of 2–3 in many cases, with larger savings over the higher bandwidth path. The transfer duration shown in the figure includes the time for the SYN exchange. These plots show that even though the overall bandwidth usage and drop rates are similar with and without Quick-Start, per-connection performance is increased when using Quick-Start.

7 Router Algorithms

This section discusses several possible Quick-Start algorithms for routers to use to choose when to approve Quick-Start requests and how much capacity should be allocated when approving requests. We start with a basic algorithm that requires minimal state, and proceed to an extreme

Quick-Start algorithm that keeps per-flow state for approved Quick-Start requests. It is desirable for routers to be able to process Quick-Start requests efficiently. At the same time, the Extreme Quick-Start algorithm explores the ability of the router to selectively approve Quick-Start requests in order to maximize the use of Quick-Start bandwidth by the end-nodes. A final consideration, that of attackers wishing to leverage Quick-Start in denial-of-service attacks, is investigated in the next section.

7.1 Basic router algorithms

Quick-Start requests represent an increased packet processing burden for routers that may also result in an increased end-to-end delay for packets with Quick-Start requests. Therefore, it is important that the algorithm for processing the Quick-Start requests at routers be as efficient as possible, with a small memory footprint.

To know if there is sufficient bandwidth available on the output link to approve a Quick-Start request, the router needs to know the raw bandwidth and have an estimate of the current utilization of the link. The router also has to remember the aggregate bandwidth approved for use by end hosts in the recent past to avoid approving too many requests and over-subscribing the available capacity. In this section we consider the algorithms used by routers to process Quick-Start requests for point-to-point links; algorithms for multi-access links are left as future work.

The first router design choice concerns the router's method for estimating the recent link utilization. There are a range of measurement and estimation algorithms from which to choose, including alternatives for the length of the measurement period. We discuss two methods for estimating the link utilization, the moving average and measuring the peak utilization. We also note that assessing alternate algorithms is an area for future work.

The **moving average** estimation technique uses a standard exponentially weighted moving average to assess the utilization over the recent past. This scheme was originally used for Quick-Start in [22]. We define $U(t)$ as the utilization at time t, $M(t)$ as the link utilization measurement at time t, δ as the interval between utilization measurements and w as the weight for the moving average. The utilization is defined as:

$$U(t+\delta) \leftarrow w * M(t+\delta) + (1-w) * U(t) \qquad (3)$$

We note that the weight w should depend on the interval δ, so that the utilization is estimated over the desired interval of time.

The **peak utilization** estimation technique records the link utilization measurements over the most recent N time intervals, and uses the highest of the N measurements as the utilization. Thus, if each time interval is s seconds, then the peak utilization method takes the peak s-second link utilization measurement over the most recent $N * s$ seconds. The peak utilization method reacts quickly to a sudden increase of link utilization, but also remembers a period of high utilization in the recent past. Unless otherwise noted, we use $N = 5$ intervals of 150 msec each.

```
avail_bw = bandwidth * (1 - utilization);
avail_bw = avail_bw - recent_qs_approvals;
approved = avail_bw * ALLOC_RATE;
if (rate_request < approved) {
  approved = rate_request;
}
recent_qs_approvals += approved;
```

Figure 7: The Share algorithm for processing Quick-Start requests.

```
util_bw = bandwidth * utilization;
util_bw = util_bw + recent_qs_approvals;
if (util_bw < qs_thresh * bandwidth) {
  // Approve Quick-Start Request
  approved =
      qs_thresh * bandwidth - util_bw;
  if (rate_request < approved) {
    approved = rate_request;
  }
  recent_qs_approvals += approved;
}
```

Figure 8: The Target algorithm for processing Quick-Start requests.

In addition to the two methods for estimating link utilization, we consider two different algorithms for deciding whether to approve a given Quick-Start request and how much capacity to grant in an approval. Both these algorithms rely on knowing *recent_qs_approvals*, the aggregate bandwidth promised in recently-approved Quick-Start requests — ideally over a time interval at least as long as typical round-trip times for the traffic on the link. If the time interval for this assessment is too small, then the router forgets recent Quick-Start approvals too quickly, and could approve too many requests, thus over-subscribing the available bandwidth. On the other hand, if the time interval is too large, the router errs on the conservative side and remembers recent Quick-Start approvals for too long. In this case the router counts some of the Quick-Start bandwidth twice, in the remembered request and also in the measured utilization, and as a result may deny subsequent Quick-Start requests unnecessarily. Unless otherwise noted, we use 150 ms as the length of *recent_qs_approvals*.

The **Share** algorithm is introduced in [22] and given in Figure 7. The algorithm uses the output link's raw bandwidth and the recent utilization estimate to allocate up to a pre-set fraction ALLOC_RATE of the unused bandwidth for each arriving request. The *rate_request* variable represents the incoming request and *approved* represents the approved rate request that will be forwarded with the packet. The *Share* algorithm does not follow the design criteria we

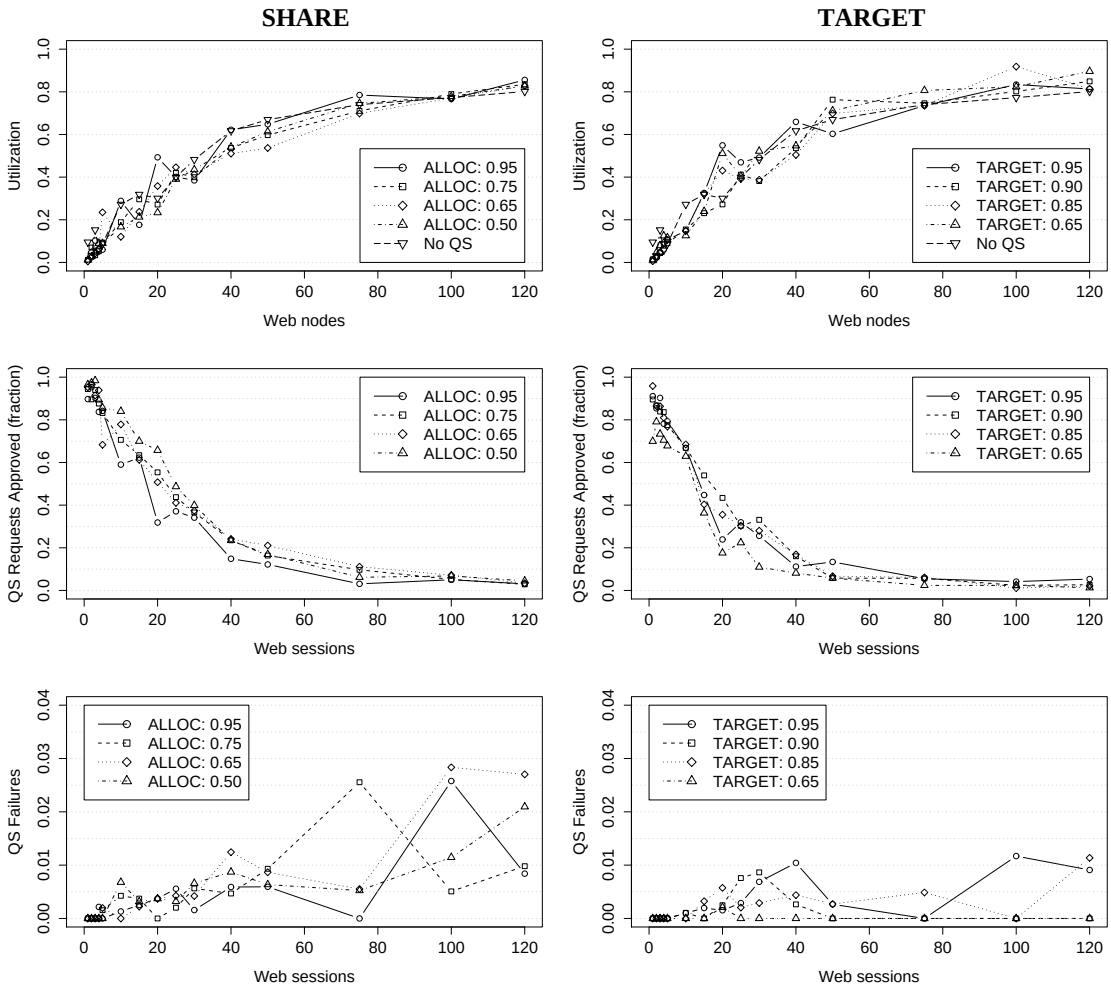

Figure 9: Comparison of *Share* and *Target* algorithms.

have sketched thus far in this paper that Quick-Start requests should only be approved when a given link is significantly under-utilized; the *Share* algorithm approves a request for up to a fixed fraction of the available bandwidth, regardless of the levels of utilization. We include an assessment of the *Share* algorithm in this paper in order to (i) compare the router algorithms we introduce with previous work and (ii) to validate our design criteria that Quick-Start should in fact only be used when all routers along a path are significantly under-utilized.

The **Target** algorithm, given in Figure 8, approves Quick-Start requests only when the link utilization, including the potential bandwidth of recently-granted Quick-Start requests, is less than some configured percentage of the link's bandwidth, denoted *qs_thresh*. This gives a router direct control over the notion of "significantly under-utilized". When a Quick-Start request is approved, the approved rate is reduced, if necessary, so that the total projected link utilization does not exceed *qs_thresh*.

Figure 9 shows simulations with the Share and Target algorithms. The simulations use a range of values for the ALLOC_RATE parameter in the *Share* algorithm and a range of values for the *qs_thresh* parameter in the *Target* algorithm. Both the Share and the Target algorithms use the peak utilization method for estimating link utilization.

The top graph of Figure 9 shows the overall link utilization for each simulation. The middle graph shows the fraction of Quick-Start Requests approved. Finally, the bottom plot shows the fraction of Quick-Start failures. The main difference between the two algorithms is that the Share algorithm approves more Quick-Start requests and experiences a larger number of Quick-Start failures than the *Target* algorithm as the network becomes more congested. We note that the ALLOC_RATE parameter does not control *whether* the Share router approves a Quick-Start request; it only controls the *size* of the approved request. The Share algorithm approves Quick-Start Requests even at high utilization levels. Even though the approved requests are for progressively smaller portions of the bandwidth the rate of failure increases. Finally, we note that the fraction of failure for both algorithms is relatively small. However, given that both algorithms have roughly the same complexity the

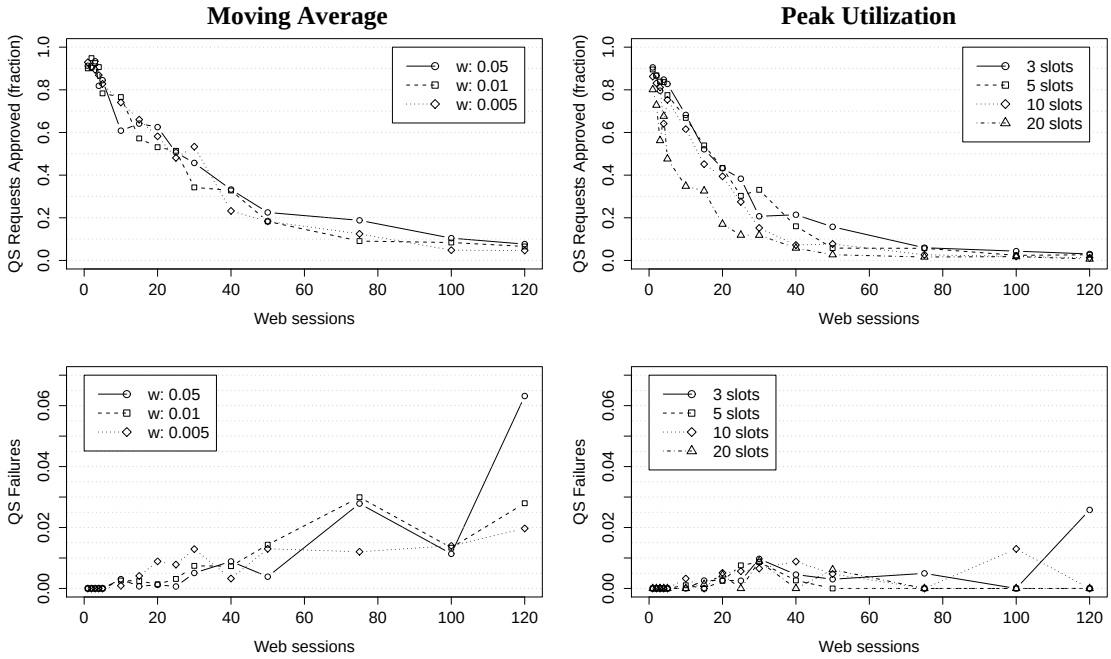

Figure 10: Comparison of moving average and peak utilization mechanisms.

Target algorithm would be preferred given the results in Figure 9.

Figure 10 compares the moving average and peak utilization methods for estimating link utilization. The simulations use the *Target* algorithm with a 10 Mbps shared link and a target level of 90%. The top graphs show the fraction of Quick-Start requests approved, and the bottom graphs show the fraction of approved Quick-Start requests with dropped packets. The moving average simulations were run with a range of values for the weight w, and the peak utilization simulations were run with a range of values for the number of 150-msec intervals over which the peak utilization was chosen. As Figure 10 shows, the method for estimating the link utilization does not significantly affect the approval rate of Quick-Start requests, but it does affect the failure rate; simulations using the moving average link utilization have a higher fraction of Quick-Start failures.

Figure 10 shows that the selection of the weight w in the moving average equation does not have a strong effect on the number of Quick-Start failures. The weight controls the time interval over which the link utilization is estimated, but the moving average method still estimates the *average* utilization; it doesn't take into account the variance of traffic intensity that can be present, particularly on links with low to moderate levels of link utilization. For Quick-Start, where the router doesn't want to approve Quick-Start requests that could result in even transient congestion, tracking the average link utilization can result in unwanted Quick-Start failures.

For the simulations with the peak utilization method, the Quick-Start failure ratio is generally lower than with the moving average method. When there are more than 50 web servers, using only three recent measurements for peak utilization causes more Quick-Start failures than when larger number of intervals are used. With twenty intervals there are hardly any Quick-Start failures. However, when ten or more intervals are used, the approval algorithm is also significantly more conservative, with fewer Quick-Start requests being approved.

7.2 Extreme Quick-Start in routers

We use the term *Extreme Quick-Start* for a Quick-Start router that maintains per-flow state about Quick-Start requests. With Extreme Quick-Start we can analyze how much Quick-Start performance could be improved if router efficiency was not a limiting factor. For example, an Extreme Quick-Start router could perform the following actions:

- A router could keep track of individual approved Quick-Start requests, and note when the Quick-Start bandwidth resulting from that request begins to arrive at the router (if in fact it does). This allows the router to more accurately estimate the potential Quick-Start bandwidth from Quick-Start requests that have been approved but not yet used at the end nodes.
- A router could keep track on the fairness of Quick-Start request approvals. If it appears that there are a number of requests that are not approved because earlier requests have allocated all of the available Quick-Start bandwidth, the router could reduce the rate approved for individual requests in order to achieve better fairness between flows.

It is useful for an Extreme Quick-Start router to know the

RTTs of flows, in order to set the length of the interval for measuring the arrival rate of packets from a flow after an approved Quick-Start request. There are a number of techniques for routers to estimate flows' RTTs [12]. In the analysis below, we assume that the Extreme Quick-Start router implements a reliable method for evaluating RTTs.

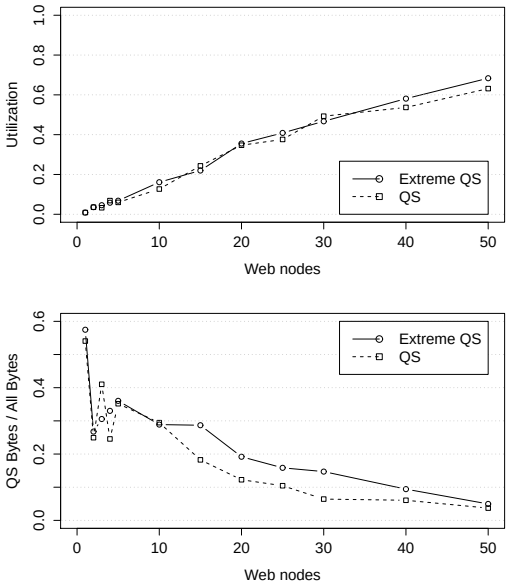

Figure 11: Basic Quick-Start and Extreme Quick-Start with a highly-tuned *recent_qs_approvals* parameter.

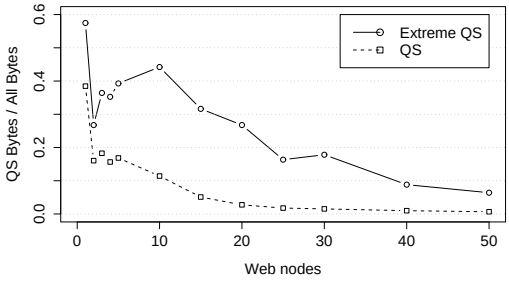

Figure 12: Basic Quick-Start and Extreme Quick-Start with a conservative *recent_qs_approvals* parameter.

Figure 11 compares the basic Quick-Start algorithm and the Extreme Quick-Start algorithm for scenarios with a small range of RTTs, with the assumption in this scenario that the RTTs are known (or easily guessed) by the router, and the router can accurately set *recent_qs_approvals* to roughly match the round-trip time. In these simulations, the basic Quick-Start variant uses the *Target* algorithm with the peak utilization method. The Extreme Quick-Start variant uses a router that keeps track of approved Quick-Start requests separately for each flow, updating its state during the transmission of the Quick-Start window as the packets arrive, and achieving a more accurate estimate of the overall amount of Quick-Start traffic that is still expected to arrive. Figure 11 shows a scenario with a range of round-trip times from 80 to 120 msec, and with the length of *recent_qs_approvals* set to 100 msec for basic Quick-Start. From the top plot we see that the utilization is nearly the same regardless of whether basic Quick-Start or Extreme Quick-Start is employed. However, the bottom figure shows that the fraction of bytes transmitted using Quick-Start is greater when Extreme Quick-Start is used by the router to track each allocation in detail. This illustrates Extreme Quick-Start's power in terms of more closely tracking resources so that more requests can be approved. This scenario is certainly not typical, but there could be some initial Quick-Start deployment scenarios, such as in limited Intranets, where there is a limited range of RTTs, and also where the traffic and network characteristics could be accurately estimated.

As a point of contrast we changed the length of *recent_qs_approvals* to 1.5 seconds to investigate Extreme Quick-Start in the context of a basic Quick-Start router that does not have a "typical" RTT and therefore chooses a conservative setting (i.e., this setting results in few Quick-Start failures, but also fewer Quick-Start request approvals). Figure 12 shows Quick-Start traffic as a fraction of the total amount of data transmitted. In this simulation we also found the utilization of basic Quick-Start and Extreme Quick-Start to be nearly identical (the plot is not shown due to space constraints). Figure 12 shows that the fraction of bytes sent during the Quick-Start phase of the connections is greater when using Extreme Quick-Start. The reason for this is that the Extreme Quick-Start router is able to keep track of the unused allocation separately for each flow as the packets arrive. Therefore, less wasted capacity is allocated by Quick-Start which allows more connections to be approved to use Quick-Start. The difference between basic Quick-Start and Extreme Quick-Start in this figure is larger than the difference shown in Figure 11 due to the more conservative setting for the length of *recent_qs_approvals*.

8 Attacks on Quick-Start

Quick-Start is vulnerable to denial-of-service attacks along two vectors: (i) increasing the router's processing and state load and (ii) causing temporary bogus allocations of Quick-Start capacity that will never be used but may prevent legitimate flows from having their Quick-Start requests approved. Since Quick-Start requests represent a processing burden on the routers involved, a storm of requests may cause a router's load to increase to the point of impacting legitimate traffic. Given the processing burden imposed by Quick-Start, this could well be worse than a simple packet flooding attack. A simple limit on the rate Quick-Start requests will be considered (with a policy of ignoring requests sent in excess of this rate) mitigates this attack on the router itself. In the case of Extreme Quick-Start another problematic aspect of a storm of packets is the memory requirement to track bogus "connections".

The second type of attack is more difficult to defend against. In this attack arbitrarily large Quick-Start requests are sent by the attacker through the network without any further data transmission. With a relatively low-rate stream of packets, this can cause a router to allocate capacity to the attacker's connections and thus temporarily reduce the amount of capacity that can be allocated to legitimate Quick-Start users. Note that the attack does not actually consume the requested bandwidth and therefore the performance of connections competing with attacks is no worse than connections that simply don't make use of Quick-Start. These attacks are particularly difficult to defend against for two reasons. First, the attack packets do not have to belong to an existing connection to do damage. And, second, since the attack just involves a Quick-Start request traversing the network path in one direction only to trigger bogus allocations, a response is not required. Therefore, spoofed source addresses are a possible aggravating factor for both hiding the location the attack is originating from and causing a simple blacklisting defense to fail.

An additional problematic aspect of Quick-Start is that legitimate requests could well cause the same impact as attack packets. Consider a Quick-Start request that is approved by the first router for some given rate, R, which the router then marks as "allocated" for some period of time. Now assume the same request hits a downstream router that either does not understand Quick-Start requests, reduces the rate to less than R or decides it cannot approve any Quick-Start request. In this case, the first router has allocated some amount of capacity that will not be used because of the conditions elsewhere in the network. From the vantage point of the first router this is similar to the attack described above. In other words, capacity allocated for Quick-Start goes unused and therefore reduces the router's ability to approve further Quick-Start requests[8].

Since Quick-Start is a loosely-connected distributed approach, routers have few options for dealing with allocations that are never used (or, not fully used). One approach is to use the notions of Extreme Quick-Start to track a host's use of Quick-Start and to disallow Quick-Start for hosts that have previously used less than their previous allocations. This approach is barely useful if an attacker can spoof source addresses because each attack packet could simply use a random source address. Further, it opens the door for another attack type — namely, that an attacker can prevent a particular host from ever using Quick-Start by making a bogus request on the victim's behalf, thereby getting the victim blacklisted. In addition, using a blacklist approach seems heavy-handed in the context of legitimate traffic that does not fully use their Quick-Start allocation (as sketched above).

[8] At first glance, allowing the router to watch the Quick-Start responses offers more information. However, due to asymmetric routing we cannot assume that a router will see the Quick-Start responses. In addition, an arbitrary router has way to tell if the *TTLDiff'* in the response is valid and therefore whether the sender will ultimately make use of the response.

Another approach is for Extreme Quick-Start routers to track the fraction of Quick-Start allocations hosts use and then make this a factor in the approval of subsequent requests. For instance, if some host requests a rate of X bytes/sec but uses only $X/2$ bytes/sec because of a downstream limitation, a router may decide to halve future rate requests from that host. An Extreme Quick-Start router has the required information to identify hosts that frequently make Quick-Start requests for more bandwidth than is actually consumed. Therefore, the Extreme Quick-Start router can reduce subsequent rate requests approved for these hosts.

We implemented the following algorithm in the Extreme Quick-Start router. The router stores both the Quick-Start allocation, $A(F)$, and the amount of bandwidth used, $B(F)$, during the Quick-Start phase for each flow, F. After the monitoring period has elapsed, the router calculates the fraction of the allocation actually consumed as $C = B(F)/A(F)$, limiting the maximum C to 1. The router maintains a score $S(H)$ for each sending host H as follows:

$$S(H) \leftarrow w*\max(C, S(H)) + (1-w)*\min(C, S(H)) \quad (4)$$

In our simulations we set the gain w to 0.2 and used a measurement interval of 1.5 seconds. Instead of a pure moving average, we selected a function that reacts quickly to hosts that often make larger requests than they end up using. When a new request arrives, the router decreases the incoming rate request by the factor $S(H)$ for the given host H.

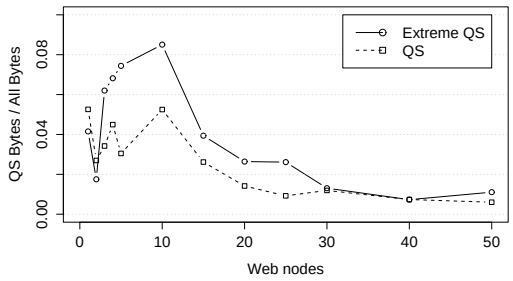

Figure 13: Impact of large Quick-Start requests for all TCP connections when accounting for abuse.

Figure 13 compares the performance of basic Quick-Start and the variant of Extreme Quick-Start sketched above. The web servers make static Quick-Start requests of 2 Mbps for all TCP connections, regardless of the object size. As the figure shows, when adjusting the allocation approved based on previous usage, Extreme Quick Start is able to allow a greater fraction of traffic to utilize Quick-Start compared to the case when the router does not track allocation usage.

Tracking per-host and per-connection state to mitigate this problem may be a high barrier. However, we note that (i) developing schemes based on aggregate traffic that do not require fine-grained tracking may be possible and (ii)

even if fine-grained tracking is required a router that is able to approve Quick-Start should be under-utilized and therefore may have some cycles to spare (and could simply turn off *all* Quick-Start activity when busy). Due to space limitations we defer an in-depth study of such schemes to future work.

9 Conclusions and Future Work

In this paper we have discussed the potential costs and benefits of Quick-Start on performance in an uncongested environment, the appropriate response to the loss or ECN-marking of a Quick-Start packet, and the range of algorithms for routers for processing Quick-Start requests. However, there are many issues we could not thoroughly study in this work, and we list some of the more significant below.

• Effectiveness: How effective would Quick-Start be in practice, in realistic scenarios of five or ten years from now? Would Quick-Start be of great benefit to users who could send an entire large transfer in a single round-trip time over an under-utilized path? Or would most of the potential Quick-Start bandwidth be "wasted" by legitimate requests denied by downstream routers, by requests from aggressive senders sending a request each round-trip time, and by malicious requests whose sole purpose is to deny Quick-Start bandwidth for other users?

• Incentives: Would routers have sufficient incentives to implement Quick-Start, considering the potential benefits, but also the additional processing costs and possible security concerns Quick-Start may introduce? Our current belief is that Quick-Start could be first deployed in networks where the routers and the end-hosts have clear mutual interest in speeding up connection startup.

• Extreme Quick-Start: What would be the minimal sufficient implementation at the routers and would there be sufficient benefit in deploying more complex algorithms in routers?

• Security: How severe are the additional security issues due to Quick-Start? What are the policing mechanisms that could be deployed in end-nodes and in routers to address these security issues?

• Non-IP queues: A router should not approve Quick-Start requests if it cannot reliably determine the link utilization all the way to the next hop. What would this mean, in practice, when there is an Ethernet switch, an ATM cloud, or some other non-IP queue between the router and the next-hop IP router?

References

[1] The Quick-Start Web Page. URL http://www.icir.org/floyd/quickstart.html.

[2] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

[4] Mark Allman. TCP Congestion Control with Appropriate Byte Counting (ABC), February 2003. RFC 3465.

[5] H. Balakrishnan and S. Seshan. The Congestion Manager, June 2001. RFC 3124.

[6] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996.

[7] Ethan Blanton, Mark Allman, Kevin Fall, and Lili Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP, April 2003. RFC 3517.

[8] R. Braden. Requirements for internet hosts – communication layers. RFC 1122, October 1989.

[9] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proc. of First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, USA, October 2002. ACM SIGCOMM.

[10] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

[11] A. Jain, S. Floyd, M. Allman, and P. Sarolahti. Quick-Start for TCP and IP. Internet-draft "draft-amit-quick-start-03.txt", September 2004. Work in progress.

[12] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM SIGCOMM Computer Communication Review*, 32(3), July 2002.

[13] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, PA, USA, August 2002.

[14] S. Kunniyur. AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections. In *Proceedings of IEEE ICC '03*, May 2003.

[15] A. Medina, M. Allman, and S. Floyd. Measuring Interactions Between Transport Protocols and Middleboxes. In *ACM SIGCOMM/USENIX Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004.

[16] Venkata Padmanabhan and Randy Katz. TCP Fast Start: A Technique For Speeding Up Web Transfers. In *Proceedings of IEEE Globecom*, November 1998.

[17] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, and J. Sterbenz. A Swifter Start for TCP. Technical Report 8339, BBN Technologies, 2002.

[18] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, November 2000.

[19] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[20] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001.

[21] E. Seurre, P. Savelli, and P.-J. Pietri. *EDGE for Mobile Internet*. Artech House, 2003.

[22] S. Sundarrajan and J. Heidemann. Study of TCP Quick-Start with NS-2. Unpublished report, University of South California, 2002.