Determining an Appropriate Sending Rate Over an Underutilized Network Path

Pasi Sarolahti, Mark Allman, and Sally Floyd,
under submission

Abstract

Determining an appropriate sending rate when beginning data transmission into a network with unknown characteristics is a fundamental issue in best-effort networks. Traditionally, the slow-start algorithm has been used to probe the network path for an appropriate sending rate. This paper provides an initial exploration of the efficacy of an alternate scheme called *Quick-Start*, which is designed to allow transport protocols to explicitly request permission from the routers along a network path to send at a higher rate than allowed by slow-start. Routers may approve, reject or reduce a sender's requested rate. Quick-Start is not a general purpose congestion control mechanism, but rather an *anti*-congestion control scheme; Quick-Start does not detect or respond to congestion, but instead, when successful, gets permission to send at a high sending rate on an underutilized path. Before deploying Quick-Start there are many questions that need answered. However, before tackling all the thorny engineering questions we need to understand whether Quick-Start provides enough benefit to even bother. Therefore, our goal in this paper is to start the process of determining the efficacy of Quick-Start, while also highlighting some of the issues that will need to be addressed to realize a working Quick-Start system.

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 competing traffic at the given time. Traditionally, TCP [21] has used a set of congestion control algorithms for determining an appropriate sending rate [11]. The rate is controlled using a congestion window (*cwnd*), which is an upper bound on the amount of unacknowledged data that can be injected into the network.

TCP's traditional method for determining the capacity of a network path with unknown characteristics is to use the *slow start* algorithm [11], which initializes *cwnd* to 1–4 segments and then increases *cwnd* exponentially during each subsequent round-trip time (RTT) of the connection. In the best case slow-start takes $log_2 N - 1$ RTTs and requires sending $N-3$ packets before reaching a *cwnd* of N packets [11]. When there is contention for resources along the network path, slow start is a reasonable procedure. However, over underutilized paths that could support large congestion windows, possibly allowing an entire data transfer to be sent in one RTT, slow start can take much time, and require much data to be transmitted before achieving the desired sending rate.

In this paper we provide an initial investigation of the efficacy of setting the initial sending rate using *Quick-Start*, a mechanism that allows a sender to advertise a desired sending rate, while the network can approve, reject or reduce the requested rate. While Quick-Start is designed to be used with a range of transport protocols, in this paper we consider its use with TCP. When using Quick-Start, a TCP sender may advertise a desire to transmit at X bytes/second in the SYN packet. Each hop along the path may (i) explicitly approve the rate request in the SYN, (ii) explicitly reject the connection's use of a higher-than-standard initial sending rate, (iii) reduce the rate from X to some X' or (iv) do nothing, which implicitly prevents the connection's use of a higher-than-standard initial sending rate. Assuming some X' arrives at the receiver, the approved rate is echoed back to the sender in the ACK of the SYN. The sender can then fairly safely transmit at X' bytes/second. If the request is rejected the sender will fall back to standard slow start. As outlined in Section 3, routers supporting Quick-Start are not required to reserve capacity promised during the Quick-Start process. Rather, routers "allocate" aggregate Quick-Start bandwidth, and this allocation is used by the router only in deciding whether to grant future Quick-Start requests. Connections are not guaranteed the capacity "allotted"— though steps are taken in the allocation process to try to make failure a rare event.

This paper makes a number of contributions, as follows. (i) We present the first, if preliminary, well-rounded evaluation of the efficacy of Quick-Start. (ii) While alternate faster-than-slow-start schemes have been proposed, Quick-Start is the first scheme to allow a large data transfer in the first round-trip time after connection set-up, explicitly involving all nodes along a network path in arriving at an explicit appropriate sending rate. (iii) We introduce the notion of *anti-congestion control*. In other words, Quick-Start only provides a quick check to determine whether a network with unknown conditions is underutilized (uncongested) and permits a large initial sending rate. Quick-Start does not attempt to control the sending rate over the lifetime of a connection, but rather yields to standard congestion control for that task. (iv) We introduce and explore the notion of rate requests for best-effort traffic. (v) Because Quick-Start is so explicit and inclusive in choosing an initial sending rate, the scheme can serve as a baseline for evaluating alternate schemes.

This paper represents only a start to the evaluation of the costs and benefits of Quick-Start. Before Quick-Start could see wide use, a variety of questions need to be answered. This paper makes some assumptions that could not be made in the real world; for example, while Section 7 briefly discusses deployment issues such as interactions with middleboxes, IP tunnels, or non-IP queues, we do not address these issues in this paper. We investigate web transfers, focusing on medium-sized flows that are shown to get the most benefit from using Quick-Start, and assume that the TCP sender is able to determine the desired sending rate for the Quick-Start request at the time when TCP connection is being established, based on the amount of data that is going to be sent. We will discuss this issue in more detail later in the paper. These assumptions are not made to minimize the required effort needed to realize a working Quick-Start system. Rather, the assumptions are used as part of the process of understanding the efficacy of Quick-Start before puzzling through the array of details that need nailed down for a Quick-Start deployment.

While our conclusion is that Quick-Start's benefits make it an attractive area for future work we are not convinced that Quick-Start would ever be feasible for the global Internet. However, many smaller (but, not small) networks that are within a single administrative domain—and therefore are not subject to the same concerns present on the global Internet—may find Quick-Start to be an attractive mechanism. For instance, [19] shows that within one particular enterprise typical network utilization is 2–3 orders of magnitude less than the raw capacity of the network and therefore Quick-Start might be useful in better using these untapped resources. Further, [2] notes that within long-delay satellite networks faster slow start is desirable.

The rest of this paper is organized as follows. Section 2 compares and contrasts Quick-Start with related work. Section 3 details the Quick-Start mechanism and discusses design issues. Section 4 describes the simulation setup used in our study, and Section 5 illustrates the potential advantages and disadvantages of Quick-Start. Section 6 discusses the handling of Quick-Start Requests by routers. Section 7 briefly highlights deployment issues, while Section 8 outlines possible vulnerabilities of Quick-Start and discusses potential mitigations to the vulnerabilities. Finally, Section 9 offers conclusions and future work.

2 Related Work

Quick-Start was first proposed in an Internet-Draft [12]. The Internet-Draft provides a protocol specification such that implementations can be built and experiments conducted. In this paper we start the process of exploring the efficacy of Quick-Start, concentrating more on the performance and algorithmic design rather than on the details of the protocol design required to implement Quick-Start.

Sundarrajan [25] added Quick-Start support to ns-2 and conducted an unpublished investigation of Quick-Start as a class project.

There have been a number of proposals for faster variants of TCP slow-start that do not use explicit feedback from routers. These mechanisms generally fall into two categories: (i) using a small volley of data packets to measure the available capacity over a network path or (ii) leveraging the capacity found by previous or concurrent connections to the same peer.

SwiftStart [20] calls for starting slow start as usual and using packet-pair [15] with the first window of data packets to estimate the bottleneck bandwidth. That estimate is then used to rapidly increase the congestion window before the second window of data is transmitted. While it is not clear how accurate an estimate would need to be to be useful, [4] suggests that using packet-pair to determine an accurate estimate of the capacity within the first part of a TCP connection is difficult. We also note that accurate bandwidth estimation has been a popular recent research topic and that the schemes to come out of this work have largely required more than a small handful of packets to obtain accurate estimates of the path capacity [22].

The second class of mechanisms to reduce the length of the slow start phase of a connection bases the increase on the assessment of the network path by concurrent or previous connections to the same peer. Assume that some TCP connection has probed the network path and is using a congestion window of X segments. The essential idea behind this class of mechanisms is that a subsequent connection which starts right after the first connection might as well leverage this information and use an initial congestion window of X segments, as well. Further, if the connections are running in parallel then the connections can share some global congestion window. TCP Fast Start [18] and the Congestion Manager [5] are examples of this class of mechanisms. Clearly, if a connection starts and there is no history about the peer this mechanism is of no benefit.

XCP (Explicit Control Protocol) [14] 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 a higher sending rate is allowed. Quick-Start also requires less new state in routers than XCP (which makes sense given the magnitude of the tasks each performs). Also, XCP faces some of the same challenges as Quick-Start (e.g., determining if all routers along some path support the given mechanism). Quick-Start can also be viewed as complimentary to XCP in that Quick-Start could be used as part of the startup phase for XCP, allowing a large initial sending rate and then transferring control to XCP. Finally, Quick-Start could provide useful data in the investigation of new, fine-grained congestion control mechanisms.

Measurement-based admission control research has investigated various algorithms at network nodes for admitting or rejecting flows, when given some Quality-of-Service requirements (see for example [8]). Quick-Start solves a somewhat similar problem regarding router algorithms for approving Quick-Start requests so that the network utilization stays within acceptable limits. However, while measurement-based admission control algorithms are designed for implementing soft Quality-of-Service based on some target parameters, such as bandwidth or packet loss rate, Quick-Start is a light-weight mechanism specifically intended for resolving the appropriate sending rate for a best-effort flow on an underutilized path.

There are several mechanisms for reserving per-connection bandwidth along a network path (e.g., RSVP [7]). Quick-Start is lighter weight in that it does not guarantee a connection a certain amount of bandwidth, and does not consider requests for bandwidth to be used over an extended period of time. However, Quick-Start tries to make sure that Quick-Start rate requests are only approved when bandwidth is actually available (e.g., failures are rare events). The Quick-Start approach is simplier than an explicit reservation system, and we believe it is more appropriate for Quick-Start's goal of rate requests for best-effort traffic in underutilized environments.

Other mechanisms for explicit congestion-related feedback from routers to end-nodes include Explicit Congestion Notifications (ECN) [23], the only current mechanism in the IP protocol for explicit congestion-related 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 [16] and VCP [26] proposals 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.

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.

3.1 Quick-Start Processing in End-Hosts

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*. The requested rate and the Quick-Start TTL are encoded in packet headers and constitute the host's request to the network. As discussed in the next subsection, the routers along the network path between the sender and receiver alter the Request, as appropriate (see Section 3.2 for details on this process). When the Quick-Start Request arrives at the transport receiver, the receiver echoes the rate request back to the sender along with *TTLDiff*$'$, the difference between the Quick-Start TTL and the IP TTL, 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 the same then the request was approved by all routers in the network path; otherwise, 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:

$$cwnd = \frac{Rate * RTT}{MSS + H} \tag{1}$$

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 in bytes 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.

One of the problems of Quick-Start is that unnecessary or unnecessarily-large Quick-Start Requests can "waste"

[1] Note that a TCP connection using Quick-Start needs to use a timer for paced transmission.

potential Quick-Start bandwidth—even though routers do not make guaranteed reservations for the "allocated" bandwidth. 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. As a result, 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 with 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 can try to determine the size of the object, and then tell TCP how many bytes will be sent; 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 determine the size of an object because the object is being read from a pipe or some live source. In Section 5.2 we illustrate the problems of not making a reasonably accurate rate request and offer some strategies for coping.

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 6 discusses algorithms that could be used by routers in approving or denying a Quick-Start request in more detail.

Finally, as we have alluded to previously, we discuss router algorithms in terms of "allocating" capacity, but 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. Because the "allocations" are not hard guarantees that require enforcement, routers implementing Quick-Start are not required to keep a burdensome amount of Quick-Start state. The required additional state at routers consists of only a handful of aggregate measurements.

4 Simulation Setup

In the following sections we use the ns-2 simulator [1] to explore Quick-Start. Unless otherwise noted, the simulations presented in the remainder of the paper use the scenario described here.

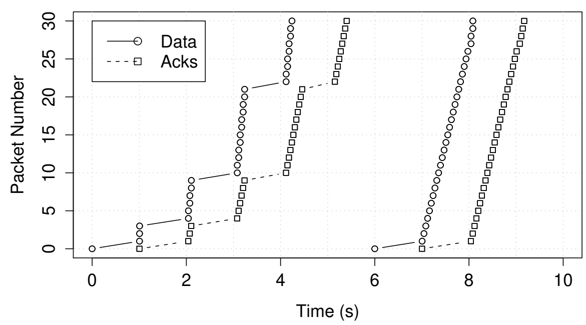
We use a network comprised of three routers, R_1–R_3, arranged in a chain. The two links between the routers each have bandwidth of L_{bw} and a one-way link delay of L_d. Unless otherwise noted, L_{bw}=10 Mbps and L_d=20 msec. The routers use drop-tail queuing with a maximum queue size of 150 packets.

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 [10]. 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 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 traffic attempts 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. The web object sizes are generated using a ParetoII distribution with an average parameter of 400 packets and 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, as Quick-Start is only effective on connections that are larger than TCP's initial window. We also ran simulations with other web traffic and network parameters, and the observations were similar as discussed in Sections 5 and 6. Our web traffic simulations are run for 150 seconds, and they were repeated 12 times (with means reported in this paper).

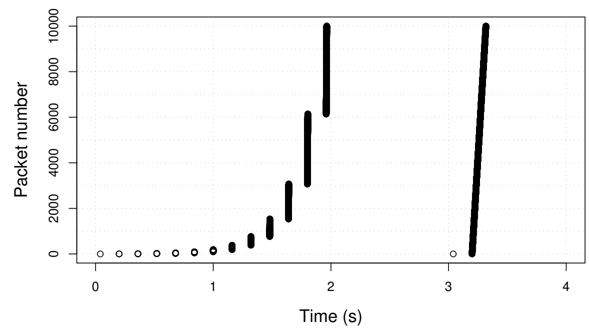
A few simulations make use of a single transfer at a time. These simulations use FTP to transfer a file of a given size over the network given above with no other traffic present.

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

Our simulation scripts will be released with the final version of the paper.



1A: a 384 Kbps link and 1-second RTT.



1B: a 10 Gbps link and 0.16-second RTT.

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

5 Connection Performance

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

5.1 Ideal Behavior

In an ideal Quick-Start scenario over an under-utilized network path, the TCP sender would be able to transmit much of its data in the initial congestion window. Figure 1 illustrates the ideal Quick-Start behavior by displaying time-sequence plots of two connections[3]. In each case, the first connection is a standard TCP connection that uses slow-start to begin transmission (with an initial *cwnd* of 3 segments after the three-way handshake). In the top graph, the second connection shows a connection where an approved Quick-Start Request allows the sender to transmit 25 of its 30 data packets in the initial window. When the first acknowledgment for data arrives at the TCP sender, the data transmission 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.

In the bottom graph, an approved Quick-Start Request for 1 Gbps in a 10Gbps network allows the TCP sender in the second connection to send all of its 10,000-packet transfer in the initial window.[4] The connection using Quick-Start completes the data transfer in one round-trip time, compared to the 12 round-trip times required by the non-Quick-Start connection. This graph shows both the potential power and potential danger of Quick-Start. On the one hand, the increase in performance is tremendous. On the other hand, the burst of traffic (even if spread over an RTT) is also tremendous and could potentially have a large impact on the network.

Figure 2 shows the performance improvement from using Quick-Start across a range of file sizes. These simulations involve a simple scenario with capacity set at 100 Mbps, various link delays, routers with unlimited buffers, routers willing to allocate 90% of their capacity to Quick-Start requests and TCP making large enough Quick-Start Requests to cover the whole link bandwidth. In each simulation, only a single flow is active. The results show that using Quick-Start aids performance — especially for medium-sized transfers that are not much larger than the approved Quick-Start request. The plot shows that Quick-Start is less beneficial for short transfers (e.g., small web objects), because the transfer time is already short without Quick-Start.

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

[3]The top scenario was motivated by a GPRS/EDGE wireless scenario [24].

[4]For clarity, the connections in these simulations do not use Limited Slow-Start [9], which can help high-bandwidth connections by limiting the number of segments by which the congestion window is increased for one window of data during slow-start.

In addition, Quick-Start's benefits drop off for long transfers, where the initial startup phase is transient and steady state behavior dictates the overall performance. These results are similar to earlier results from Sundarrajan [25]. In general, the optimal Quick-Start behavior occurs with a transfer of N packets, and an initial congestion window from Quick-Start of N packets as well. In this case, a data transfer of $log_2(N+2)-1$ round-trip times (with an initial window of two packets) is reduced to a data transfer of a single round-trip time.

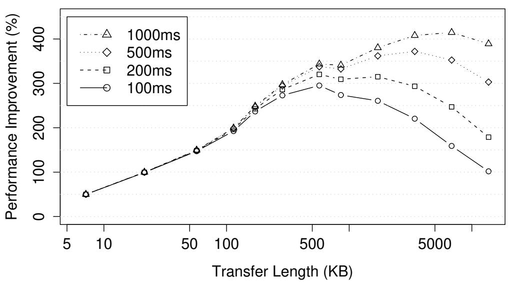


Figure 2: Relative improvement with Quick-Start, for a single flow over a 100 Mbps link, with a range of propagation delays.

5.2 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. An ideal Quick-Start request would contain the precise sending rate the connection could use. However, determining 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 sent.

To illustrate the problems caused by overly large Quick-Start requests, we simulate two scenarios of web traffic, where a new TCP connection is used for each web object transferred. In the Greedy scenario, all TCP connections use a static Quick-Start request of 2 MB/sec. In contrast, in the Ideal scenario, which is admittedly unrealistic, each request is optimal for the amount of data its connection has to transmit. In addition, Quick-Start is not used in the Ideal scenario if the connection is able to send all data in the standard three-segment initial *cwnd*. The simulations use an average web object size of 60 packets.

In the Greedy scenario, because all connections use a large, fixed-size Rate Request requests are generally granted for only the first connection in each web session. The router is generally unable to approve requests of later connections in each session, 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. In this scenario, 9% of Quick-Start requests are approved and 220 KBps of data is transmitted during Quick-Start. In the Ideal scenario connections use ideal sizes for their Rate Requests and requests are approved more often since there are fewer wasted approvals. For the Ideal scenario, 40% of Quick-Start requests are approved and 769 KBps are transmitted during Quick-Start, showing the increased overall effectiveness of appropriately-sized Quick-Start requests.

While the Ideal scenario 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[5], C bytes/sec, requests could be chosen to be no larger than C. Going even further, large web servers could make policy decisions to disallow a single TCP connection from requesting more than some fraction of the access link bandwidth in a Quick-Start request. In addition, a sender could take into account the size of the local socket buffer, S bytes, and the receiver's advertised window, W bytes, when choosing a request size[6]. Given an RTT of R sec,[7] TCP can send no faster than $min\ (S, W)\ /\ R$ bytes/sec (assuming W is non-zero and using S otherwise). Finally, and more speculatively, if an application informs 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, with a request of $(O+(O/MSS)*H)/R$ bytes/sec for a given MSS size and estimated header size of H bytes. In our simulations TCP sender uses this method to determine the size of the Quick-Start request. While these techniques do not necessarily provide for an ideal Quick-Start request, they could well provide a more reasonable request than simply picking a static rate for all cases.

When a 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 miscalculates the rate that should be approved. After a Quick-Start failure, the TCP sender disregards the *cwnd* determined using Quick-Start, and uses slow-start to open *cwnd* just as would have happened without Quick-Start.

[5] A number of operating systems and applications already ask users to configure such information (at least in broad terms) and so this does not seem like an onerous expectation.

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

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

5.3 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 requests; 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 6 we study methods for routers for choosing whether to approve Quick-Start requests, and how much capacity to grant each request. We also illustrate the implications of using Quick-Start when the router is not significantly under-utilized.

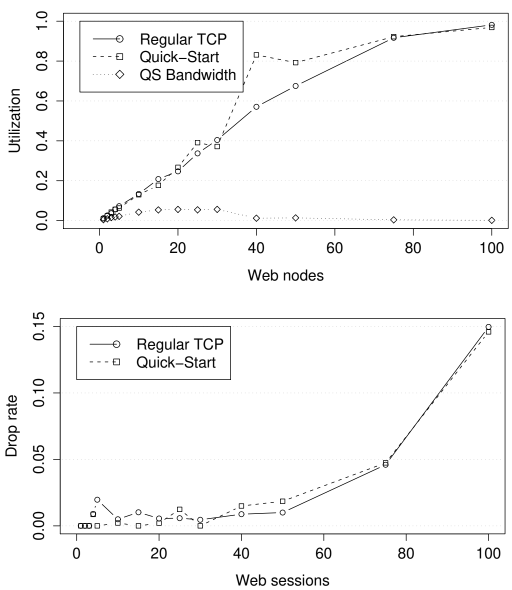


Figure 3: Comparison of utilization and drop rates with and without Quick-Start, with a 10 Mbps shared link.

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 affected. 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, an alternate traffic model is possible, where the successful use of Quick-Start would increase the amount of data sent and received by each user. For example, users could have a fixed amount of time available for using the network, rather than 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 3 shows the overall utilization and aggregate drop rates with and without Quick-Start, as a function of traffic load on the 10 Mbps shared link. For each web session, there are also ten ftp tranfers of a hundred packets each, starting at random times. This traffic mix was chosen to give many large Quick-Start requests, as something of a worst-case scenario, to increase the chances of finding a scenario where Quick-Start packets interfere with the throughput or loss rates of other traffic on the link. As shown in the figure, the link 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 3 shows the bandwidth used by Quick-Start packets in the simulations using Quick-Start — indicating that Quick-Start is in fact being used in the scenarios with less overall traffic. For the scenario shown, the web traffic generator uses a ParetoII distribution with an average parameter of 60 packets for web object size.

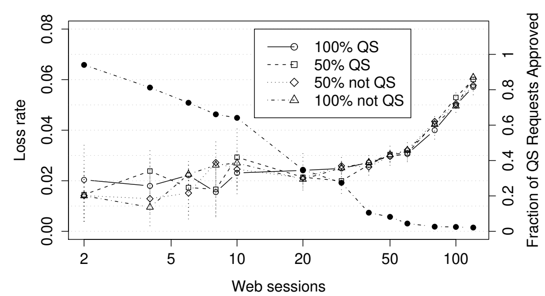


Figure 4: Comparison of drop rates of regular TCP flows when half of the flows either has Quick-Start enabled or disabled, with a 10 Mbps shared link.

Figure 4 shows packet loss rates for a scenario using only web traffic, for the following three simulations: (i) all TCP flows use Quick-Start, (ii) 50 % of the TCP flows use Quick-Start and (iii) none of the flows use Quick-Start. For simulation (ii) the plot shows the drop rate for the Quick-Start and non-Quick-Start flows separately. Additionally, the graph shows the fraction of approved Quick-Start requests for simulation (i) to give a feel for the actual Quick-Start usage. The figure shows that the use of Quick-Start does not have a significant effect on the packet loss rates regardless of the amount of traffic attempting to use Quick-Start. The packet loss rates have a clearly increasing trend as the number of web sessions is increased. In addition, as the loss rates increase we note that the likelihood of Quick-Start requests being approved decreases (as expected, since Quick-Start is to be used in non-congested networks). Based on these simulations, Quick-Start does not seem to be harmful to competing traffic in the network (regardless of whether the competing traffic uses Quick-Start).

Figure 5 shows per-connection performance of all traffic in a simulation with three 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 that 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, the use of Quick-Start increases per-connection performance.

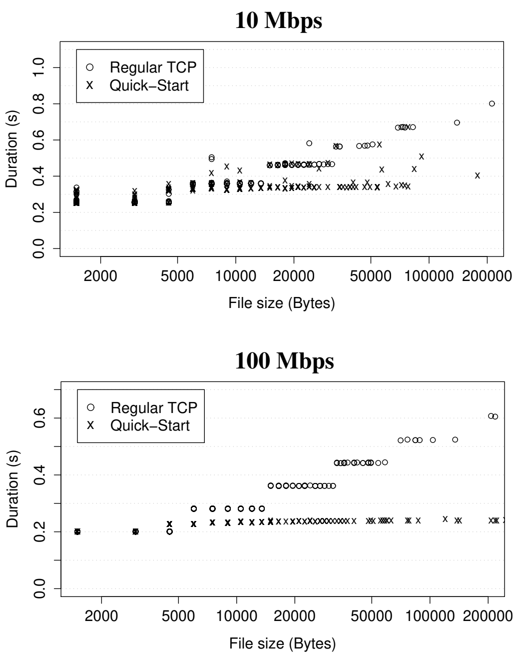


Figure 5: Per-connection performance with and without Quick-Start, with 10 Mbps and 100 Mbps shared links and three web sessions.

6 Router Algorithms

This section discusses several possible Quick-Start algorithms for routers to use when considering Quick-Start 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 an *ideal router* to selectively approve Quick-Start requests in order to maximize the use of Quick-Start bandwidth by the end-nodes. Extreme Quick-Start is introduced as a point of comparison and not as a proposal for the way routers should handle Quick-Start.

6.1 Basic router algorithms

Quick-Start requests represent an increased packet processing burden for routers, and this could 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. That is, the router has to keep a small amount of new state about the aggregate traffic (and, no per-flow state). 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. Developing and 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 [25]. 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:

$$U(t+\delta) \leftarrow w * M(t+\delta) + (1-w) * U(t) \qquad (2)$$

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

With **peak utilization** estimation, the router measures the link utilization over the most recent N time intervals, and takes the highest of the N measurements as the current link 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

```
util_bw = bandwidth * utilization;
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 6: The Target algorithm for processing Quick-Start requests.

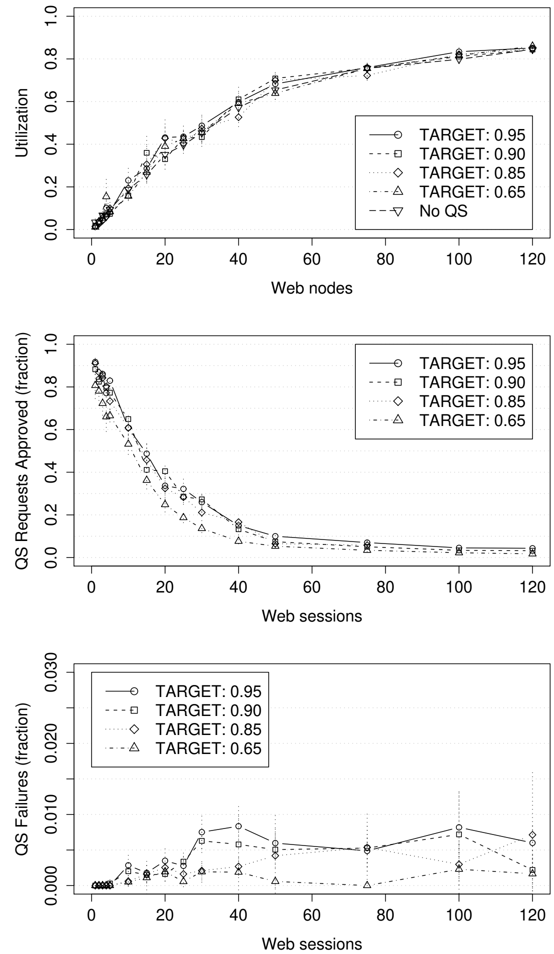


Figure 7: Evaluation of Target algorithm.

of link utilization, but also remembers a period of high utilization in the recent past. Unless otherwise noted, we use $N = 5$ intervals with interval length of $s = 150$ msec which covers most of the RTTs in our simulated network.

In addition to the two methods for estimating link utilization, we consider how to decide whether to approve a given Quick-Start request and how much capacity to grant in an approval. This process relies 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 compute *recent_qs_approvals* as the aggregate Quick-Start bandwidth approved in the most recent two 150-ms intervals, including the current interval.

The **Target** algorithm, given in Figure 6, approves Quick-Start requests only when the link utilization, including the potential bandwidth use 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 7 shows simulations with the Target algorithm. The simulations use a range of values for the *qs_thresh* parameter in the Target algorithm. In these simulations, the Target algorithm uses the peak utilization method for estimating link utilization. The top graph of Figure 7 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. We note that the fraction of failures for the Target algorithm is relatively small (less than 1% in all cases tested).

Figure 8 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 *qs_thresh* 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 N of 150-msec intervals over which the peak utilization was chosen. The legend in each figure shows the overall time interval for the estimation; for the moving average graph, this is estimated as the time needed for $-1/ln(1-w)$ measurements, where a measurement is taken for each departure from the queue [27].

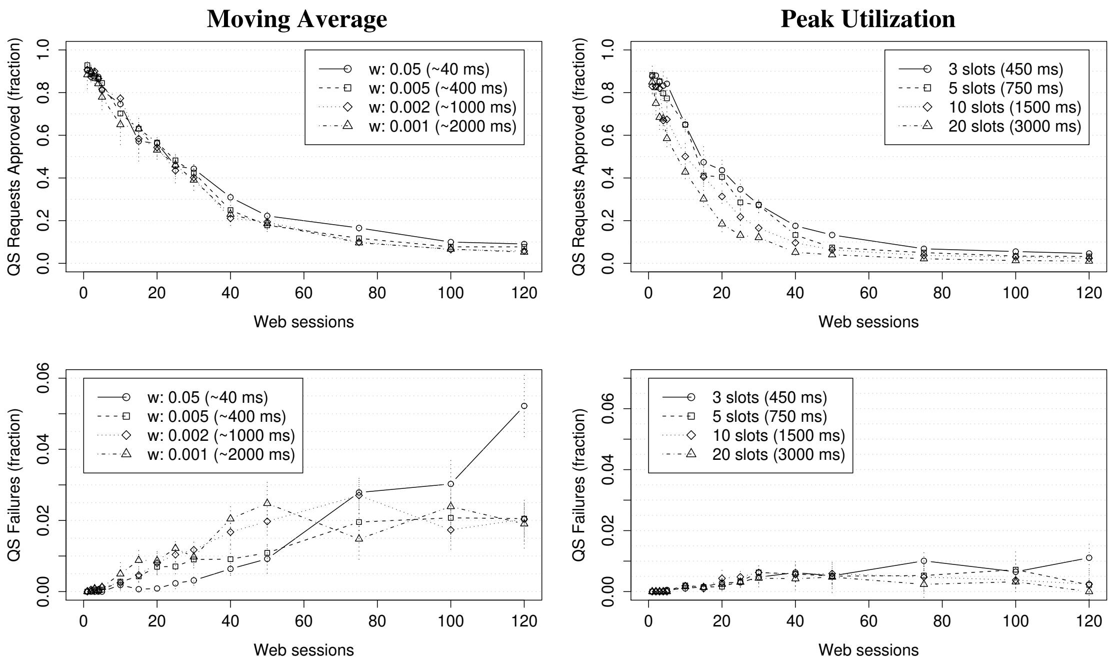


Figure 8: Comparison of moving average and peak utilization mechanisms.

As Figure 8 shows, the approval rate of Quick-Start requests can be slightly higher with the moving average method, but the failure rate is higher also, regardless of the value for the weight w. The weight controls the time interval over which the link utilization is estimated, but the moving average method still estimates the *average* utilization. The moving average does not take into account the variance of traffic intensity that can be present. A router that does not want even transient congestion should not estimate the average link utilization since this will likely lead to Quick-Start failures.

For the simulations with the peak utilization method, on the other hand, the Quick-Start failure ratio is generally lower than with the moving average method. However, the performance is sensitive to the number N of intervals used. Larger values of N lead to lower acceptance rates, but also to lower failure rates in congested environments. This illustrates a potentially tricky balancing act in determining the larger time period over which link utilization is measured, and in determining the interval for assessing peak utilization within the larger time period.

6.2 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, and the term *Basic Quick-Start* for a Quick-Start router that does not maintain per-flow state, but follows the algorithms in the section above. While not necessarily realistic in practice, 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 [13]. In the analysis below, we assume that the Extreme Quick-Start router implements a reliable method for evaluating RTTs.

For each flow, an Extreme Quick-Start router estimates the number of bytes expected to arrive in the Quick-Start phase, based on the approved rate request and the estimated RTT. The Extreme Quick-Start router also maintains the number of received bytes for each flow. From this information the router can compose a detailed estimate of currently unused Quick-Start bandwidth, and therefore is able

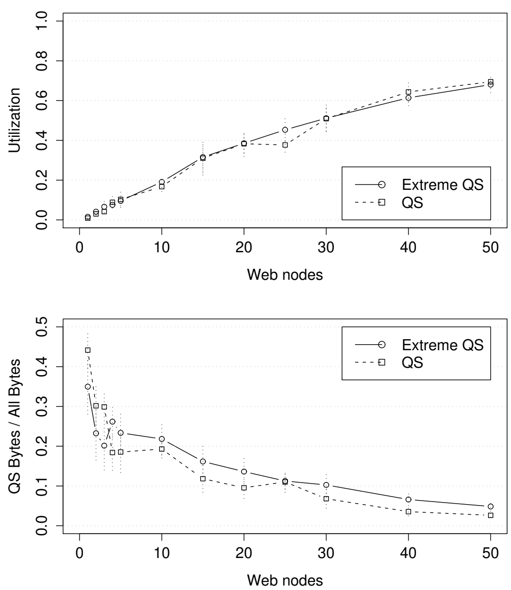


Figure 9: Basic Quick-Start and Extreme Quick-Start with a highly-tuned *recent_qs_approvals* parameter.

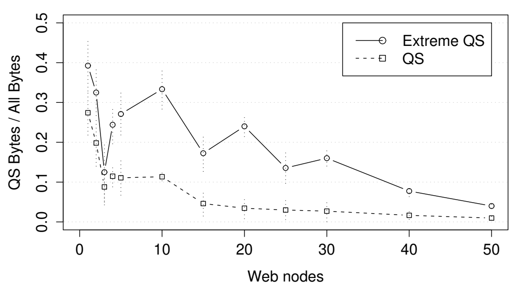


Figure 10: Basic Quick-Start and Extreme Quick-Start with a conservative *recent_qs_approvals* parameter.

to more accurately establish how much bandwidth is available for new rate requests. As Basic Quick-Start does not track per-flow state but only maintains aggregate information, Basic Quick-Start is more conservative in its estimation of the available bandwidth. After the initial window of data has arrived at a router, there is a period of time where some data is counted twice; *recent_qs_approvals* accounts for bandwidth that has been promised for Quick-Start requests, while the packets that have arrived are also accounted for in the link utilization. Extreme Quick-Start aims to remove this overlap, resulting in both a higher acceptance rate for the Quick-Start requests, and approvals of higher Quick-Start bandwidth.

We use two examples to illustrate the difference between Basic Quick-Start and Extreme Quick-Start. Figure 9 compares Basic Quick-Start and Extreme Quick-Start for scenarios with a small range of RTTs (80–120 msec), 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 (100 msec). In these simulations, Basic Quick-Start uses the Target algorithm with the peak utilization method. From the top plot we see that the link 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 slightly greater when Extreme Quick-Start is used by the router to track each allocation in detail. 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 configured. The figure shows that in such conditions, with carefully tuned parameters, it is possible to achieve nearly the same performance with basic Quick-Start as with Extreme Quick-Start.

As a point of contrast we changed the computation of *recent_qs_approvals* to include the most recent two 1.5-second intervals, 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 10 shows Quick-Start traffic as a fraction of the total amount of data transmitted. The figure shows that the fraction of bytes sent during the Quick-Start phase of the connections is greater when using Extreme Quick-Start. This illustrates Extreme Quick-Start's power in terms of more closely tracking resources so that more requests can be approved. Therefore, Quick-Start involves less wasted capacity, allowing more Quick-Start requests to be approved. The difference between basic Quick-Start and Extreme Quick-Start in this figure is larger than the difference shown in Figure 9 due to the more conservative setting for the length of *recent_qs_approvals*. In this simulation the link utilization with Basic Quick-Start and Extreme Quick-Start was also nearly identical.

7 Deployment Issues

The previous sections have shown that Quick-Start has some potential to increase performance without significantly impacting competing traffic. We next turn our attention to several practical issues that must be addressed before a working Quick-Start system could be realized. Although we discuss the issues from Quick-Start's perspective, many of the issues are more broadly applicable.

Chickens and Eggs. Quick-Start is only of use when it is supported by both end systems and all the routers along the path. This leads to the "chicken-and-egg" deployment problem, that there is little incentive to being the first node to deploy Quick-Start. Because of the incremental-deployment problems, we expect that initial deployments of Quick-Start would happen within networks or Intranets with centralized control, where hosts and routers both have an interest in aiding performance.

Interactions with Middleboxes. There are middleboxes in the current network that drop packets containing known or unknown IP options [17]. This could cause delays for connections using Quick-Start, as packets containing Quick-Start requests would have to be retransmitted without the request. Again, 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.

Non-IP Queues. A further deployment issue concerns the possibility of non-IP queues along a path. A router should not approve Quick-Start requests if it cannot reliably determine the link utilization all the way to the next IP hop. What this would mean, in practice, when there is an Ethernet switch, an ATM cloud, or other non-IP queue between the IP router and the next-hop IP router is left as future work.

Tunnels. IP tunnels are a challenge for a mechanism that requires processing at every router. Some tunnel implementations that do not know about Quick-Start might encapsulate a packet without decrementing the inner IP TTL field first at the tunnel ingress. As a result, a seemingly-valid Quick-Start Request with an unchanged *TTLDiff* is carried in the inner header, while the outer header most likely does not carry a Quick-Start Rate Request. If the tunnel egress decapsulates the packet without modifying the inner IP TTL field or otherwise rejecting Quick-Start, it is possible that the Quick-Start Request would be falsely approved. This problem would be shared by any protocol that requires processing at every router (e.g., XCP), and also presents a consideration in the design of future tunnel protocols.

The difficulties of incremental deployment and the problems of middleboxes, coupled with the potential problem of attacks on Quick-Start bandwidth discussed in Section 8, suggest that Quick-Start could remain in controlled networks for quite some time, where the incremental-deployment barriers are reduced, the range of middleboxes is under more control, and attack traffic can more easily be monitored and controlled. In addition, in such a controlled environment, it is likely that all of the routers along a path would support Quick-Start, reducing the problem of Quick-Start requests that are denied simply because of routers that are not Quick-Start capable. It is even possible that Quick-Start would remain a mechanism largely for use in controlled environments, and would never see ubiquitous deployment in the global Internet.

8 Attacks on Quick-Start

8.1 Threats

Quick-Start is vulnerable to denial-of-service attacks along two vectors: (i) increasing the router's processing and state load and (ii) using bogus Quick-Start requests to temporarily reduce the available Quick-Start bandwidth. Since Quick-Start requests represent a potential processing burden for routers, a storm of requests may cause a router's load to increase enough to affect 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 at which Quick-Start requests are 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, an attack on the available Quick-Start bandwidth, 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 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 competing connections is no worse than connections that simply don't make use of Quick-Start. Hoever, 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 problem with Quick-Start is that legitimate requests could well cause the same impact as attack packets. Consider a Quick-Start request for rate R that is approved, and therefore considered "allocated", by the first router in the path. Now assume the same request hits a downstream router that reduces the rate to some R' less than R (maybe even to zero) for whatever reason. In this case, the first router has allocated some amount of Quick-Start capacity that cannot be given to subsequent Quick-Start users 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 that capacity allocated for Quick-Start goes unused, while the router's ability to approve further

Quick-Start requests is reduced.[8] One possible use of Extreme Quick-Start to allow routers to reduce Quick-Start requests from senders that have in the past used only a fraction of their approved Quick-Start bandwidth.

In addition to Denial of Service attacks, a simple implementation of Quick-Start could be vulnerable to cheating by routers or by end-nodes. Non-conformant routers or hosts might try to modify Quick-Start messages to benefit particular connections. For instance, a receiver could 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. Mitigations for these and other attacks are discussed in the next section. We also note that such cheating would risk hurting instead of helping performance; lying about the size of the approved rate request could end up causing packet drops for Quick-Start packets, resulting in a slow-start for the connection in question.

8.2 Mitigations

In some sense, a number of the problems described above are fundamental to a lightweight system that does not require authentication of requests or per-flow state at all nodes in the network path. For instance, when a router observes a SYN packet with a Rate Request, how is that router to know if this is a spoofed packet or a legitimate request to establish a connection with a larger-than-standard sending rate?

A first mechanism to mitigate the problems might be for senders to advertise their sending rate during the round-trip time after a valid Quick-Start request. With a small amount of per-flow state, this could allow routers to adjust their notion of the amount of Quick-Start capacity that has been "allocated". In other words, if a flow requested and was approved for R_1 bps at a given router and then advertised some R_2 bps as their sending rate, the router could decrease its record of "allocated" Quick-Start bandwidth by $R_1 - R_2$. This would mitigate the problem of overly large requests consuming Quick-Start resources they will not be able to use due to downstream limits.

Another possible addition would be of a "two pass" structure. In this scheme, a first request would be sent as usual. Assuming a valid rate R is returned, the sender could then send a second request for rate R through the network for verification (and tagged as such). During this second pass the routers could not to reduce the rate, but could reject the use of Quick-Start for the flow. Also, during this second pass the routers could change a "provisional" allocation into a "confirmed" allocation. As above, this mechanism could be used to reduce the problem of downstream rate reductions that invalidate an upstream router's estimate of allocated Quick-Start bandwidth. In addition, this mechanism would reduce the impact of spoofing senders; if the rate given in the second pass is larger than the rate approved by the router from the first pass then the request will not be confirmed by the router, and the router could update its estimate of allocated Quick-Start bandwidth. A malicious, non-spoofing sender would still be able to request Quick-Start bandwidth without using it. However, this is a more tractable case since a non-spoofed sender would be identifiable, and therefore policy could be applied to its traffic.

Finally, a nonce can be used to catch receivers trying to game Quick-Start. Suppose that the rate in each request is encoded in N bits in the packet header, allowing for $2^N - 1$ rates to be encoded. Now, suppose a nonce field of length $X \times (2^N - 1)$ is included in the request and initialized to a random value. For each decrement of the rate from Y to $Y - 1$, a particular X-bit portion of the nonce would be overwritten by a random value. As an example, a 4-bit encoding of the rate request could take on 15 non-zero rates. A minimum sized nonce would be 15 bits in length. When a router decremented the request from 15 to 14, the router would set the first bit in the nonce field to a random value; similarly, a router decrementing the request from 14 to 12 would set the second and third bits of the nonce to random values. The receiver would echo back the nonce to the sender in its reply to the rate request. The sender would then be able to verify that the reported rate request corresponded to the unchanged portions of the nonce. The nonce would largely prevent receivers from lying about the rate that arrived. Even if the receiver knows the original rate request (which is not a given), the chances of the receiver correctly guessing the original nonce to "prove" that the rate was not reduced below that in the network would be $\frac{1}{2^X} \times S$ for a rate that was reduced S steps in the network.

None of the above mechanisms remove the fundamental tension between having a lightweight scheme to determine if a network path can support an increased sending rate on the one hand, and having a scheme that is immune from malicious behavior on the other. However, some combination of these schemes may well offer enough mitigation to make Quick-Start practical in some production networks (even if not in the Internet itself). The particulars of making the specific engineering tradeoffs to design these mechanisms are left as future work.

[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 no way to tell if the *TTLDiff′* in the response is valid and therefore whether the sender will ultimately make use of the response.

9 Conclusions and Future Work

In this paper, we explore a mechanism for *anti-congestion control*, where the task is not to detect and respond to congestion, but to determine when the sender can use a higher sending rate than it would otherwise. We present the first well-rounded study of Quick-Start, and show that with only minimal additional router state and processing and an additional request upon connection setup, transfer times for medium-sized files can be reduced significantly in an uncongested network. While Quick-Start can aid per-connection performance, it does not lead to higher drop rates in the network, because Quick-Start requests are only approved when the network is underutilized. Thus, while Quick-Start can help users in an underutilized network, it should have little or no effect in a congested network.

We have also explored the downsides of Quick-Start, including thorny deployment considerations and security problems. We have sketched potential mitigations to some of these problems in this paper, but much additional design and experimentation will be required before Quick-Start will be useful in the global Internet (if it ever will be). However, Quick-Start may be of use on networks under the control of a single organization, which could benefit from Quick-Start while at the same time shedding some of the thorny problems (e.g., security threats) presented when multiple administrative domains come into play.

While this paper only considers the use of Quick-Start in determining a connection's *initial* sending rate, another fruitful area of work is to explore the use of Quick-Start after idle periods or mobility events, when a connection is significantly under-utilizing the network path or has no understanding of the path's characteristics. Other areas of future work are to consider the use of Quick-Start with other transport protocols, and to explore in more detail algorithms for setting the size of Quick-Start requests at end-nodes and processing Quick-Start requests at routers. We expect other issues for future work to also arise with the experimental deployment of Quick-Start in small controlled networks.

Acknowledgements

Amit Jain first presented the Quick-Start idea. Srikanth Sundarrajan developed the initial Quick-Start implementation for ns2.

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.

References

[1] NS Simulator. URL http://www.isi.edu/nsnam/ns/.

[2] M. Allman, S. Dawkins, D. Glover, J. Griner, J. Heidemann, T. Henderson, H. Kruse, S. Ostermann, K. Scott, J. Semke, J. Touch, and D. Tran. Ongoing TCP Research Related to Satellites, Feb. 2000. RFC 2760.

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

[4] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *SIGCOMM '99*, Sept. 1999.

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

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

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, Sept. 1997.

[8] L. Breslau, S. Jamin, and S. Shenker. Comments on the Performance of Measurement-Based Admission Control Algorithms. In *Infocom 2000*, Mar. 2000.

[9] S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. RFC 3742, Mar. 2004.

[10] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *HotNets-I*, Oct. 2002.

[11] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM '88*, Aug. 1988.

[12] A. Jain, S. Floyd, M. Allman, and P. Sarolahti. Quick-Start for TCP and IP. Internet-draft "draft-ietf-tsvwg-quickstart-01.txt", Oct. 2005. Work in progress.

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

[14] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *SIGCOMM 2002*, Aug. 2002.

[15] S. Keshav. A Control-Theoretic Approach to Flow Control. In *SIGCOMM '91*, pages 3–15, Sept. 1991.

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

[17] A. Medina, M. Allman, and S. Floyd. Measuring Interactions Between Transport Protocols and Middleboxes. In *SIGCOMM/USENIX Internet Measurement Conference*, Oct. 2004.

[18] V. Padmanabhan and R. Katz. TCP Fast Start: A Technique For Speeding Up Web Transfers. In *IEEE Globecom*, Nov. 1998.

[19] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A First Look at Modern Enterprise Traffic. In *SIGCOMM/USENIX Internet Measurement Conference*, Oct. 2005.

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

[21] J. Postel. Transmission Control Protocol. RFC 793, Sept. 1981.

[22] R. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network*, November/December 2005.

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

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

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

[26] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One More Bit Is Enough. In *SIGCOMM 2005*, Aug. 2005.

[27] P. Young. In *Recursive Estimation and Time-Series Analysis*, pages 60–65, 1984.