An Evaluation of TCP with Larger Initial Windows

Mark Allman
NASA Lewis Research Center/Sterling Software
mallman@lerc.nasa.gov

Chris Hayes
Lucent Technologies
chayes@lucent.com

Shawn Ostermann*
School of Electrical Engineering and Computer Science
Ohio University
ostermann@cs.ohiou.edu

Abstract

TCP's slow start algorithm gradually increases the amount of data a sender injects into the network, which prevents the sender from overwhelming the network with an inappropriately large burst of traffic. However, the slow start algorithm can make poor use of the available bandwidth for transfers which are small compared to the bandwidth-delay product of the link, such as file transfers up to few thousand characters over satellite links or even transfers of several hundred bytes over local area networks. This paper evaluates a proposed performance enhancement that raises the initial window used by TCP from 1 MSS-sized segment to roughly 4 KB. The paper evaluates the impact of using larger initial windows on TCP transfers over both the shared Internet and dialup modem links.

1 Introduction

TCP [Pos81b] uses a *congestion window* (*cwnd*) to control the amount of unacknowledged data the sender injects into the network, and the slow start algorithm to gradually increase the value of *cwnd* [JK88]. The slow start algorithm is employed at the beginning of a transfer and after loss is detected by the expiration of TCP's retransmission timer [JK88] [Ste97]. In addition, many TCP implementations use the slow start algorithm after a connection has been idle for a certain amount of time (1 round-trip time in most BSD-derived implementations). This paper only investigates the impact of increasing the initial value of *cwnd* at connection startup.

This paper does not attempt to answer all questions about using a larger initial window. The impact on the connection using the larger initial window is studied. However, the impact on competing TCP connections and overall network congestion is not studied. The implications of using a larger initial window on competing traffic in a relatively simple simulated topology is examined in [PN98]. The competing traffic for these tests is assumed to be using an initial window of 1 or 2 segments[1]. An interesting, yet extremely difficult, test is to measure the impact of using larger initial windows in a large network when all hosts are using the same initial window size.

*Supported in part by NASA's Lewis Research Center Satellite Networks and Architectures Branch.

[1] A well documented and wide-spread bug in some TCP implementations causes the use of a 2 segment initial window in certain situations [PAD+98] [All97a].

The remainder of this paper is organized as follows. Section 2 briefly outlines the slow start algorithm. Section 3 describes the proposed change in the initial window. Section 4 describes our testing framework. Section 5 outlines the experimental setup used in this investigation. Section 6 presents the results of tests using larger initial windows over a dialup channel, as well as over the shared Internet. Finally, section 7 presents our conclusions.

2 Standard Slow Start

When a gateway receives segments faster than it can forward them onto an outgoing link, the segments are queued for later transmission. When a gateway exhausts its queue memory, incoming segments are discarded. The slow start algorithm prevents a sender from overwhelming intervening gateways with an inappropriately large burst of data, that may cause segment loss. However, slow start can make poor use of available network capacity, particularly for transfers which are small compared to the bandwidth-delay product of the link [Kru95] [AHKO97] [All97b] [Hay97].

As originally defined by Jacobson [JK88], slow start initializes *cwnd* to 1 segment[2]. For each acknowledgment (ACK) the sender receives during slow start, the value of *cwnd* is incremented by 1 segment. This algorithm increases the size of *cwnd* exponentially until its value reaches the receiver's advertised window or the sender detects network congestion. Assuming the receiver ACKs each incoming segment and no segments or ACKs are lost, the amount of time needed for *cwnd* to reach the advertised window size is given in equation 1 [JK88].

$$slow\ start\ time = R\ log_2 W_A \tag{1}$$

In this equation, R is the round-trip time (RTT) of the given network path and W_A is the size of the receiver's advertised window (in segments).

[2] In practice, *cwnd* is usually stored in terms of bytes. For simplicity, we discuss the value of *cwnd* in terms of segments in this paper.

RFC 1122 [Bra89] defines the use of an optional *delayed acknowledgment* mechanism. Receivers implementing delayed acknowledgments are not required to send an ACK for every incoming segment. However, an ACK must be sent for every second full-sized segment that arrives. Furthermore, if a second full-sized segment does not arrive within a given timeout, an ACK must be transmitted. This timeout can be no more than 500 ms and is roughly 200 ms in most BSD-derived systems. As outlined above, the slow start algorithm increases the value of *cwnd* by 1 segment for each ACK received. Therefore, by reducing the number of ACKs returned to the sender by roughly half, the time required for *cwnd* to reach the advertised window is increased twofold. Equation 2 shows the approximate[3] amount of time required to fully open *cwnd* when the receiver employs delayed ACKs [PS97].

$$slow\ start\ time \approx 2R\ log_2 W_A \tag{2}$$

While delayed ACKs hinder slow start, the mechanism does provide advantages. The reduction in segments (ACKs) injected into the network signifies a reduction in resources and processing required by the end points as well as the intermediate gateways. Paxson [Pax97] found that delayed ACKs are common in many TCP implementations popular today. The resources saved by employing delayed ACKs can have throughput advantages for bulk transfers in some environments [Joh95]. However, using delayed ACKs can introduce a performance problem with an initial window of 1 segment. In this situation, the sender transmits a single segment and waits for the corresponding ACK. A single segment arriving at the receiver does not generate an immediate ACK. Rather, the receiver will wait for a second segment or the delayed ACK timer before transmitting an ACK. When the sender's *cwnd* is 1 segment, the sender will be forced to wait

[3] It is difficult to *exactly* quantify the time required to open *cwnd* in the face of delayed ACKs because of the delayed ACK timer. The timer implementation, the length of the timeout and the RTT between the sender and receiver all interact to make prediction of the exact increase difficult, and beyond the scope of this paper.

(sending no new data and wasting time) until the delayed ACK timer expires.

3 Larger Initial Windows

A recent proposal [AFP98] suggests increasing TCP's initial window (IW) from 1 segment to the value given in equation 3.

$$IW = min\ (4*MSS,\ max\ (2*MSS,\ 4380)) \quad (3)$$

For example, if the maximum segment size (MSS) is 1024 bytes, the initial window consists of 4 segments, each of size 1024 bytes, as shown in equation 4.

$$\begin{aligned} IW &= min\ (4*MSS, \\ &\quad max\ (2*MSS,\ 4380)) \\ &= min\ (4*1024, \\ &\quad max\ (2*1024,\ 4380)) \\ &= min\ (4096,\ max\ (2048,\ 4380)) \\ &= min\ (4096,\ 4380) \\ &= 4096\ (or,\ 4\ segments) \end{aligned} \quad (4)$$

As outlined in [AFP98], the larger initial window is used at the beginning of a transfer and after an idle period. The *cwnd* continues to be reduced to 1 segment following the expiration of the retransmission timer (RTO) as originally outlined in [JK88].

There are several advantages of using a larger initial window. First, increasing the initial window reduces the amount of time required to open *cwnd* to the receiver's advertised window. Specifically, the time required by slow start to increase the value of *cwnd* from an initial size of W_I to an advertised window of W_A when the receiver ACKs every incoming packet is given in equation 5.

$$slow\ start\ time = R(log_2 W_A - log_2 W_I) \quad (5)$$

Likewise, equation 6 gives an approximation of the time required to increase the window from an initial window size of W_I to an advertised window of W_A when the receiver generates delayed ACKs.

$$slow\ start\ time \approx 2R(log_2 W_A - log_2 W_I) \quad (6)$$

As these equations show, the time required to reach a window of size W_A is reduced by the number of RTTs needed to reach an advertised window of W_I. The time saved by this change can be significant for transfers that are short compared to the *delay*bandwidth* product of the network. The benefit for bulk transfers is less dramatic, because as the length of the transfer increases, the impact of slow start on the total performance decreases.

Finally, using an initial window of more than 1 segment reduces the transfer time by 1 delayed ACK timeout when the receiver implements delayed ACKs. As outlined in section 2, when using a 1 segment initial window the receiver must wait for the delayed ACK timeout to expire before sending an ACK, since a second segment never arrives. Therefore, the total savings provided by the proposed initial window size is up to 3 RTTs plus a delayed ACK timeout [AFP98].

While increasing the size of the initial window can benefit performance, the change can also have disadvantages. In a highly congested environment, using a larger initial window may increase the amount of loss experienced by a TCP connection. This additional loss may reduce performance. The use of a larger initial window may also cause additional loss in competing TCP connections sharing a highly congested bottleneck. Therefore, in some circumstances, using a larger initial window may hurt performance and be unfair to competing traffic.

4 Preliminary Tests

To verify that the findings of the experiments discussed in this paper would be applicable to the general Internet, we conducted extensive preliminary tests to understand the characteristics and dynamics of the networks involved. This section discusses these preliminary experiments.

4.1 Remote Hosts

In the Internet experiments presented in this paper, the sending machine was a Pentium-Pro based computer running NetBSD 1.2.1 at NASA's Lewis Research Center (LeRC). The changes we made to the NetBSD TCP implementation to use larger initial windows are outlined in appendix A. We chose remote Internet sites randomly from logs collected by the Ohio University Computer Science WWW server. The sample of hosts used in this paper consists of the first 100 hosts randomly chosen from the WWW log that met the following two criteria.

1. The host was running a TCP discard server.
2. The host advertised a window that exceeded our largest initial window (32 segments, or 16 KB)[4]. This ensured our initial window was not constrained by the receiver's advertised window.

Appendix B provides details about the path characteristics between LeRC and the remote hosts used in our tests.

4.2 Characterizing the Common Path

By running tests to a large group of Internet sites, we hoped to test a cross-section of Internet paths to determine the impact of using a larger initial window. Since the sender is at LeRC for all transfers, the test results could have been skewed by nearby network problems. For instance, a congested or underbuffered gateway near LeRC may skew the results. In this case, the majority of the transfers may be subject to the same bottleneck and therefore roughly the same behavior would have been observed over the majority of the network paths chosen.

To ensure all close gateways were neither generally congested nor underbuffered, we attempted to characterize the portion of the network path common to the majority of remote hosts in our sample. To determine the *common path* between LeRC and the sample of remote hosts, we used the *traceroute*[5] utility to measure the route to each of the 100 remote hosts. A gateway is considered to be a part of the *common path* if it meets the following two conditions.

1. The gateway in question is the first gateway traversed, or the previous gateway was a member of the common path.
2. The gateway was in the path to at least a third of the remote hosts. Or, it was apparent that multiple routers were sharing the load at a given hop and together they were in the path to at least a third of the remote hosts.

After determining the common path used to reach the sample of remote sites, this path was probed to detect whether bottlenecks were present. We designed the *bprobe*[6] utility to measure the *burst capacity* of the common path. We define the burst capacity as the number of segments that can be sent back-to-back into the network and successfully arrive at the destination. A burst of 100 ICMP echo-request segments of size 512 bytes was injected into the network. In each of the segments transmitted, the IP time-to-live (TTL) field [Pos81a] was set to one more than the number of gateways in the common path. This forced each of the gateways in the common path to attempt to queue and forward each segment in the burst. The first gateway beyond the common path should decrement the TTL to 0 and return an ICMP time-exceeded message. The number of time-exceeded messages returned to the sender is reported by *bprobe*. We do not believe this accurately measures the burst capacity of the network, but rather reports a lower bound on the burst capacity. Several situations may cause this estimate to be conservative, including lost time-exceeded messages and routers prematurely discarding ICMP messages.

The route to each of the 100 remote Internet hosts was measured and two distinct common paths, each consisting of 7 gateways, were found. The two common paths differed only in hop 5.

[4] In this paper, 1 KB = 1024 bytes

[5] *traceroute* can be obtained from `http://www-nrg.ee.lbl.gov/`

[6] We intend to release *bprobe* in late-1998.

It appears that two gateways were sharing the load at this hop. Therefore, when probing the common path, one probe was sent through each gateway at hop 5. For the remainder of the paper the *common path* will refer to both paths, unless otherwise noted.

Since the TCP tests spanned 24 hour periods for multiple days, we measured the burst capacity of the common path for the 24 hours prior to the start of our TCP tests. The burst capacity was measured every 2 minutes in order to minimize the amount of network capacity used and to ensure that the burst tests did not interfere with one another. On average this probing period injected 427 bytes/second into the network. We believe this overall traffic rate is low enough that the burst probing has minimal impact on the network. Figure 1 shows the cumulative distribution function for the measured burst capacity. As indicated by the point on graph, roughly 15% of the time, the burst capacity of the common path was less than 32 segments (the largest initial window size tested in our experiments). From these measurements, we conclude that the burst capacity of the common path to the remote sites used in our TCP tests does not contain a bottleneck that skews the test results.

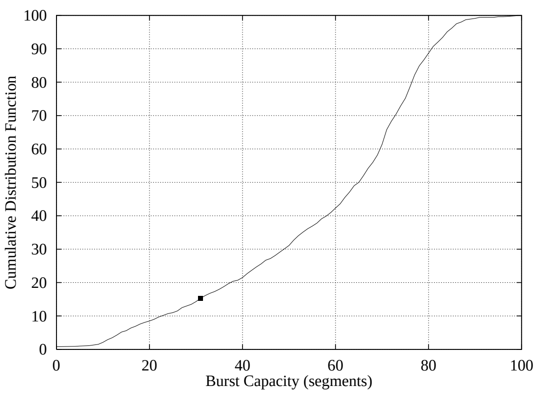


Figure 1: Burst capacity of the network in the 24 hours prior to the TCP tests presented in this paper.

5 Experimental Setup

5.1 TCP Tests

TCP transfers using various initial window sizes were employed to measure the relative costs and benefits of using larger initial TCP windows over a dialup channel and the global Internet. 16 KB transfers were used to isolate the differences in the startup behavior caused by various initial window sizes. The segment size of the sending host was 512 bytes. The tests were divided into 30 *test rounds* for statistical purposes. Each test round consists of one *test group* for each of the remote hosts in the sample (1 remote host was used in the dialup channel tests and 100 remote hosts were used in the Internet tests). A test group is comprised of one transfer using each initial window size tested to a given remote host.

5.2 Procedure

In addition to TCP transfers using various initial window sizes, the network path and burst capacity of the common path were also measured periodically during the Internet tests. These measurements were taken to ensure that any changes in the common path or general burst capacity of the common path during the TCP tests were noted. The burst capacity of the common path was determined 3 times during each test round (following test groups 33, 66 and 99; roughly every 90 minutes). At the end of each test round (approximately every 4.5 hours), the network path to each of the remote hosts was recorded using *traceroute*.

5.3 Analysis

All TCP connections were traced by a machine on the same local area network as the data sender using *tcpdump*[7]. The *tcptrace*[8] utility was used to gather statistics about the transfers. The data analysis consisted of comparing TCP transfers within each test group. For example, a transfer

[7] *tcpdump* is available from `http://www-nrg.ee.lbl.gov/`

[8] *tcptrace* is available from `http://jarok.cs.ohiou.edu/`

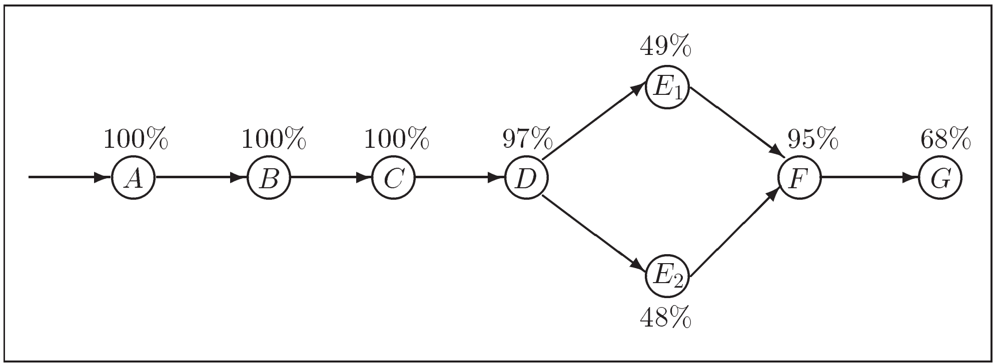


Figure 2: Measured common path from NASA LeRC to the sample of remote hosts with percentage of hosts using each gateway.

utilizing an initial window of 2 segments is compared to the transfer using an initial window of 1 segment in the same test group. The metrics we used to compare the transfers using various initial windows were transfer time and number of retransmitted segments. The differences in these metrics within each test group were averaged in the results presented in this paper.

6 Experimental Results

6.1 Common Path

As outlined in section 5.2, the route to each of the 100 remote Internet hosts was determined after each test round using *traceroute*. These measurements were then combined to determine the common path, as outlined in section 4.2. Figure 2 illustrates the gateways in the common path and the percentage of route measurements using the given gateway. Our measurements did not show a change in the common path over the course of the TCP tests. However, this does not necessarily mean that the routes did not change between measurements, as has been discussed in [Pax96]. The common path measured during the TCP tests was the same as the common path measured in the preliminary tests (as outlined in section 4.2). No gateway beyond the common path was traversed in more than 25% of the measurements.

6.2 Burst Probing

As discussed in section 5.2, the *bprobe* utility was used to determine the burst capacity of the common path 3 times during each test round. Figure 3 shows the cumulative distribution function of the measured burst capacity of the common path during the TCP tests. In only one of the *bprobe* measurements was the measured burst capacity less than the largest window size used in our TCP tests. In this case, the burst capacity was measured as 30 segments. These measurements were taken less frequently than the preliminary measurements outlined in section 4.2 (approximately every 90 minutes, as opposed to every 2 minutes in the preliminary tests). Therefore, we believe the measurements shown in figure 1 are more representative of true network conditions than those taken during the TCP tests (figure 3). However, the burst probes done in conjunction with the TCP transfers show that the common path did not develop long-term congestion that would skew the results of the TCP tests.

6.3 Modem Results

The first set of TCP tests involved a 28.8 kbps dialup modem connection. Both the sender and receiver in these tests were NetBSD 1.2.1 machines. A CSLIP [Jac90] connection was established through a Xyplex MaxServer 800 terminal server and the local phone company. Figure 4(a) shows the average percentage of time saved as a function of the initial window size when compared to a transfer using a 1 segment initial win-

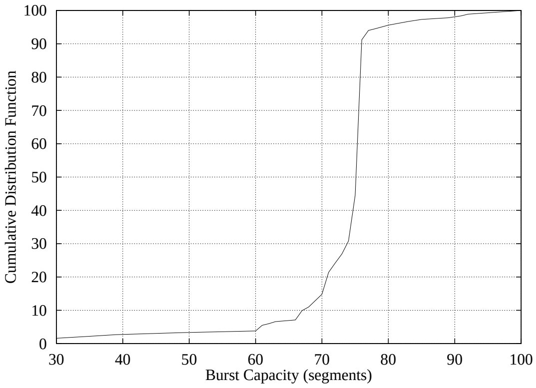


Figure 3: Burst capacity of the network during the TCP tests presented in this paper.

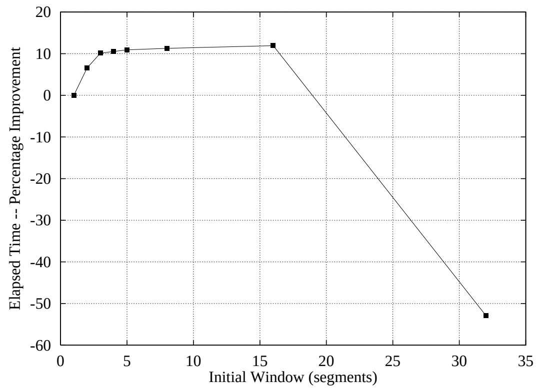
dow. Using an initial window of between 3 and 16 segments reduced transfer time by roughly 10% when compared to using an initial window of 1 segment. Figure 4(b) shows that initial windows up to 16 segments experienced no more retransmissions than did transfers using an initial window of 1 segment. However, this figure shows that when using an initial window of 32 segments, additional loss is experienced. Figure 4(a) shows the additional loss caused by using an initial window of 32 segments increased the transfer time by roughly 50%. Using the initial window suggested in [AFP98], transfer time was reduced by roughly 10% without increasing the number of retransmissions experienced over the dialup channel.

6.4 Internet Tests

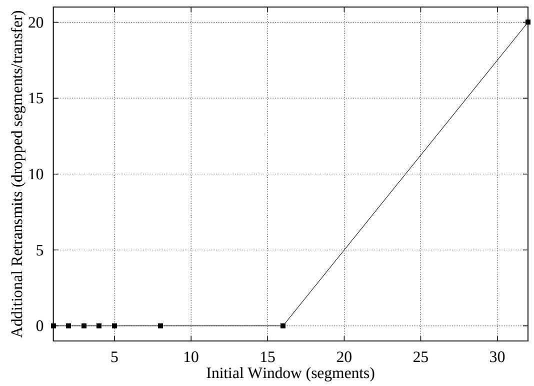
Figure 5(a) shows that as the initial window was increased, the time required to transfer a 16 KB file over the Internet was reduced. Figure 5(b) shows the increase in retransmitted segments per transfer as a function of the initial window size. Using a 4 segment initial window reduced the transfer time by roughly 25% and each transfer experienced roughly 0.04 additional retransmitted segments. The Internet tests show that 74% of transfers using an initial window of 1 segment (i.e., standard TCP) experienced no loss. Therefore, when figure 5(b) indicates that a given initial window size increased the number of retransmitted segments by 1 segment per transfer, that *usually* indicates that the transfer in question retransmitted a single segment.

Figure 5(b) shows a reduction in *retransmitted* segments when using an initial window of 4 segments, when compared to initial windows of 2 and 3 segments. When a single segment is lost from an initial window of 4 segments, the *fast retransmit* algorithm can repair the loss without waiting for a retransmission timeout (RTO) and without sending needless retransmissions. When loss is detected via the RTO timer, TCP uses slow start to retransmit segments which can cause needless retransmits (when the receiver already has the segment) [FF96]. When losing a segment from an initial window of 2 segments, the fast retransmit algorithm is not used due to a lack of returning ACKs. When a single segment is lost from an initial window of 3 segments, the fast retransmit algorithm can be used in some cases. Therefore, of the range of initial windows (2 to 4 segments) proposed in [AFP98], an initial window of 4 segments provides the best chance of recovering from a single dropped segment without unnecessary retransmissions.

Figure 5(a) exhibits a counter-intuitive result that requires further explanation. Figure 5(a) shows roughly the same throughput improvement for both 16 and 32 segment initial windows. However, discounting connection setup (which does not vary with the size of the initial window) and assuming no loss, transferring 16 KB of data would require 2 RTTs with a 16 segment initial window and only 1 RTT with a 32 segment initial window. One might reasonably expect that because the 32 segment initial window test could send the data in 1 less RTT than the 16 segment initial window test, the 32 segment test would also reduce transfer time when compared to the 16 segment test. The answer to the anomaly is found in figure 5(b). Figure 5(b) shows that using an initial window of 32 segments caused nearly 3 segments per transfer to be retransmitted on average, whereas the 16 segment test caused just over 1 retransmitted segment. We believe that this extra retransmitted segment between the 16 and 32 segment tests inserts an extra RTT into the 32 segment test, which causes both transfers to take the same



(a) Transfer time improvement when compared to standard TCP.



(b) Increase in dropped segments when compared to standard TCP.

Figure 4: Modem Results

amount of time on average and therefore have the same throughput improvement.

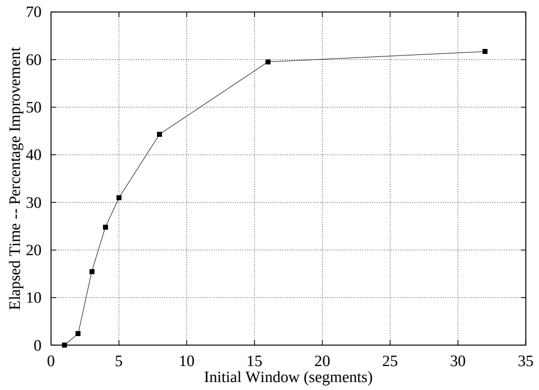
The extra RTT inserted due to the extra retransmission can be explained as follows. In the absence of selective acknowledgments (SACK) [MMFR96], the fast retransmit algorithm can only repair a single lost segment per window of data. Therefore, when multiple segments are lost from a window of data, the slow start algorithm is used to retransmit additional lost segments after the retransmission timer (RTO) expires. Figure 5(b) shows that, for the 16 segment test, more than 1 segment is retransmitted on average; the first retransmitted segment is sent using the fast retransmit algorithm and the second retransmitted segment is sent using slow start after an RTO, adding 1 RTT and 1 RTO timeout to a loss-free transfer time.

However, the 32 segment test caused almost 3 retransmits on average (from figure 5(b)). As with the 16 segment test, the first retransmitted segment is sent using the fast retransmit algorithm. In the 32 segment case, however, the slow start algorithm is used to retransmit 2 additional segments, rather than 1 as in the 16 segment test. Because slow start begins by sending a single segment and waiting for an ACK, retransmitting 2 segments requires 2 RTTs, plus the RTO timeout. Therefore, recovery takes an additional RTT when the initial window is 32 segments when compared to an initial window of 16 segments. This nullifies the RTT saved by using the larger initial window and causes the transfer time improvement to be roughly the same in both cases.

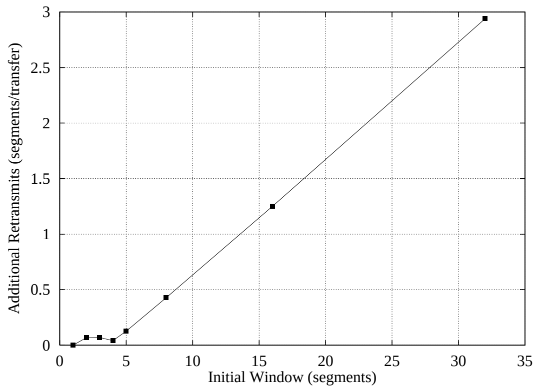
7 Conclusions

This paper evaluates the effect on TCP performance of changing the initial window size. We have found that an initial window of 2 to 4 segments, as suggested by [AFP98], decreases the transfer time of short transfers over dialup channels and the Internet. Furthermore, the change does not significantly increase the number of retransmitted segments. In addition, using an initial window of 4 segments provides a better chance for good loss recovery when compared to 2 and 3 segment initial windows.

Because re-starting an idle connection is fundamentally very similar to starting a new connection, we believe the investigation presented in this paper implicitly covers the initial window used to re-start an idle connection. However, it should be noted that this is an active research area and other methods of re-starting an idle TCP connection may prove more beneficial (e.g., rate-based pacing [VH97]).



(a) Transfer time improvement when compared to standard TCP.



(b) Increase in dropped segments when compared to standard TCP.

Figure 5: Internet Results

Acknowledgments

This work has benefited from discussions with a large number of people. First, we thank Fred Baker, Sally Floyd, Vern Paxson and Allyn Romanow for discussions involving the use of a larger initial window. Doug Dechow, Matt Mathis and Tim Shepard provided valuable feedback about our testing and analysis procedures. In addition, we thank Paul Mallasch, the IETF's TCP Implementation Working Group and the anonymous reviewers for useful feedback on our experiments and this paper. Finally, thanks to Boris Friedman and Eric Helvey for providing the dialup channel used in our experiments.

A Modified TCP Implementation

We modified a NetBSD 1.2.1 kernel to use a larger initial congestion window as described in section 3. The changes allow the initial window to be easily controlled using the *netconfig*[9] utility in our testing scripts. Two new variables have been added to the kernel as follows:

- **Initial Window Byte Limit** (`initwin_bytes`)
  When non-zero, `initwin_bytes` is an upper bound on the number of bytes allowed in the initial window. TCP sends an initial burst of `initwin_bytes` bytes in the appropriate number of MSS-sized data segments.

- **Initial Window Segment Limit** (`initwin_segs`)
  When non-zero, `initwin_segs` sets an upper bound on the number of segments allowed in the initial window. TCP sends an initial burst of `initwin_segs` segments, each 1 MSS in length.

When both `initwin_bytes` and `initwin_segs` are set, the lesser of the two values is used as the initial window. For example, if the MSS is 512 bytes, `initwin_bytes` is 4096 bytes and `initwin_segs` is 4, then the initial window will be 4 segments with a total of 2048 bytes. Consider a second example when the MSS is 1024 bytes, `initwin_bytes` is 4096 bytes, and `initwin_segs` is 5. In this case, the initial window will be 4 segments with a total of 4096 bytes.

If both `initwin_bytes` and `initwin_segs` are set to zero, the standard NetBSD 1.2.1 code is used to set the initial window. If one of the vari-

[9] *netconfig* is available at `http://www-csl.ucsd.edu/`

ables is non-zero while the other is zero, the initial window will be governed by only the non-zero variable. For instance, if `initwin_segs` is set to 5 and `initwin_bytes` is set to zero, the initial window will consist of 5 MSS-sized segments.

Most BSD-derived TCP implementations contain a bug that effectively sets the initial value of *cwnd* to 2 segments when TCP connections are opened passively [All97a] [PAD+98]. In the tests presented in this paper, all TCP connections were opened actively by the host transmitting the data. Therefore, this bug does not affect the results presented in this paper.

B Statistics about the Remote Hosts

As outlined in section 4.1, the remote Internet sites used in the TCP transfers presented in this paper were chosen randomly from a WWW server log file. Table 1 lists the distribution of domains covered by our sample of remote hosts. As the table shows, at least half the sites are outside the United States. However, the sites in domains such as `.com` and `.edu` cannot be classified one way or another. According to the Internet Domain Survey [Wiz98], 28% of hosts belong to the `.com` domain, while only 13% belong to the `.edu` domain. This is clearly not the case in our sample, as 35% of the sample belongs to the `.edu` domain and 6% of the sample belongs to the `.com` domain. We believe the discrepancy can be explained by the likelihood of hosts in the `.com` domain being located behind firewalls. These firewalls prevented the probe used to find acceptable hosts from reaching hosts inside the given organization. Even though the distribution of top-level domains is skewed in our sample, the hosts used offered a variety of network paths.

Figure 6 gives the distribution of hop counts between the sending host and the remote Internet hosts in our sample. The route to each host may have changed during the course of the tests [Pax96]. Therefore, the average number of hops measured to each remote host was used to generate the graph. We did not alter *traceroute*'s default limit of 30 hops to a given host. However, only 1 host averaged 30 hops and therefore, this limit did not skew the results.

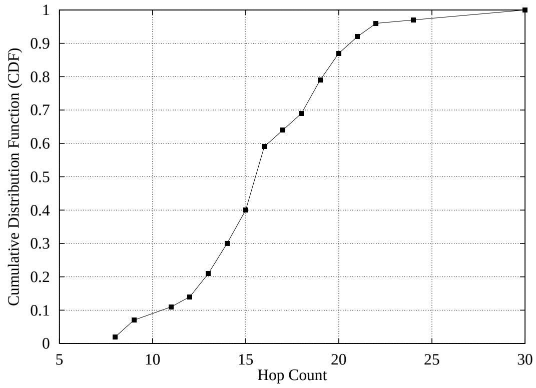


Figure 6: Hop-Count Distribution

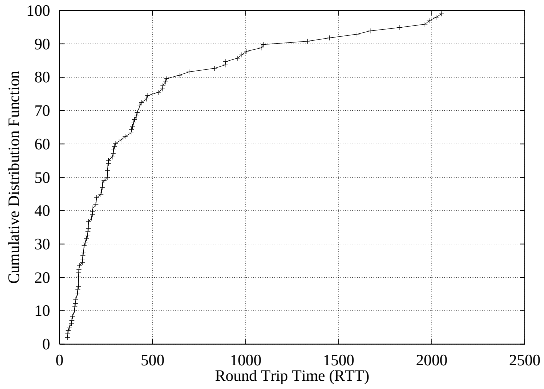


Figure 7: RTT Distribution

Figure 7 reports the distributions of RTTs between the TCP sender and the remote Internet hosts. The average RTT between the sender and each remote host was found by analyzing the trace file of the TCP transfers with *tcptrace*. The reported RTT is an average of the measured RTTs. This figure shows that the RTT to the remote hosts in our sample was sufficiently well distributed between close and far away hosts.

References

[AFP98] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window, April 1998. Internet-Draft draft-floyd-incr-init-win-03.txt (work in progress).

Domain	Frequency	Domain	Frequency
edu	35	tr	1
unknown	7	su	1
uk	7	se	1
de	7	nz	1
com	6	my	1
fr	5	mx	1
gov	4	it	1
nl	3	il	1
mil	3	id	1
kr	3	gr	1
ca	3	fi	1
au	2	es	1
tw	1	cl	1
ch	1		

Table 1: Domain Distribution

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, and Shawn Ostermann. TCP Performance Over Satellite Links. In *Proceedings of the 5th International Conference on Telecommunication Systems*, March 1997.

[All97a] Mark Allman. Fixing Two BSD TCP Bugs. Technical Report CR-204151, NASA Lewis Research Center, October 1997.

[All97b] Mark Allman. Improving TCP Performance Over Satellite Channels. Master's thesis, Ohio University, June 1997.

[Bra89] Robert Braden. Requirements for Internet Hosts – Communication Layers, October 1989. RFC 1122.

[FF96] Kevin Fall and Sally Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[Hay97] Chris Hayes. Analyzing the Performance of New TCP Extensions Over Satellite Links. Master's thesis, Ohio University, August 1997.

[Jac90] Van Jacobson. Compressing TCP/IP Headers For Low-Speed Serial Links, February 1990. RFC 1144.

[JK88] Van Jacobson and Michael Karels. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[Joh95] Stacy Johnson. Increasing TCP Throughput by Using an Extended Acknowledgement Interval. Master's thesis, Ohio University, June 1995.

[Kru95] Hans Kruse. Performance Of Common Data Communications Protocols Over Long Delay Links: An Experimental Examination. In *3rd International Conference on Telecommunication Systems Modeling and Design*, 1995.

[MMFR96] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. RFC 2018.

[PAD+98] Vern Paxson, Mark Allman, Scott Dawson, Ian Heavens, and Bernie Volz. Known TCP Implementation

Problems, March 1998. Internet-Draft draft-ietf-tcpimpl-prob-03.txt (work in progress).

[Pax96] Vern Paxson. End-to-End Routing Behavior in the Internet. In *ACM SIGCOMM*, August 1996.

[Pax97] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM*, September 1997.

[PN98] Kedarnath Poduri and Kathleen Nichols. Simulation Studies of Increased Initial TCP Window Size, June 1998. Internet-Draft draft-tcpimpl-poduri-01.txt.

[Pos81a] Jon Postel. Internet Protocol, September 1981. RFC 791.

[Pos81b] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

[PS97] Craig Partridge and Tim Shepard. TCP/IP Performance Over Satellite Links. *IEEE Network*, 11(5), September/October 1997.

[Ste97] W. Richard Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. RFC 2001.

[VH97] Vikram Visweswaraiah and John Heidemann. Improving Restart of Idle TCP Connections. Technical Report 97-661, University of Southern California, August 1997.

[Wiz98] Network Wizards. Internet Domain Survey, January 1998. http://www.nw.com.