A Web Server's View of the Transport Layer*

Mark Allman
NASA Glenn Research Center/BBN Technologies
21000 Brookpark Rd. MS 54-2
Cleveland, Ohio 44135
mallman@grc.nasa.gov

Abstract

This paper presents observations of traffic to and from a particular World-Wide Web server over the course a year and a half. This paper presents a longitudinal look at various network path properties, as well as the implementation status of various protocol options and mechanisms. In particular, this paper considers how World-Wide Web clients utilize TCP connections to transfer web data; the deployment of various TCP and HTTP options; the range of round-trip times observed in the network; packet sizes used for WWW transfers; the implications of the measured advertised window sizes; and the impact of using larger initial congestion window sizes. These properties/mechanisms and their implications are explored. An additional goal of this paper is to provide information to help researchers better simulate and emulate realistic networks.

1 Introduction

This paper presents observations of traffic to and from a particular World-Wide Web (WWW) server over the course of 17 months. This paper has several goals. First, we attempt to evaluate the performance impact and the deployment status of several features of network stacks as used in the Internet today. Second, we attempt to determine what protocol extensions or features might be useful in the future, based on the observed traffic. Finally, we hope the data presented in this paper will be useful for researchers designing simulations of Internet traffic in answering key questions about the values of key parameters (e.g., What is a realistic value for TCP's advertised window for web clients?).

World-Wide Web traffic uses the HyperText Transfer Protocol (HTTP) [FGM+97] application protocol to transfer data from web servers to user's browsers. HTTP uses the Transmission Control Protocol (TCP) [Pos81] as its transport protocol to ensure reliable delivery (to the extent possible [SP00]). The web server we observed is at NASA's Glenn Research Center (GRC). The server provides unofficial web pages for several Internet Engineering Task Force (IETF) working groups (PILC, TCP-IMPL, TCPSAT). These pages provide mailing list archives, meeting minutes, draft documents, etc. to the community. The web server is also used by several researchers for personal web pages, as well as small, project-specific pages.

This study is mainly focused on the characteristics of the client's networking stacks and the network paths between the clients and our web server. Therefore, our results could be biased by some feature of the server we chose, the underlying operating system or the network on which the server is located. Several alternative measurement methodologies could have been used, as follows.

- We could have monitored the WWW traffic traversing a link closer to the center of the network. This would have produced traces of a large number of network paths with many endpoints. However, tracing network connections in the middle of the network may make the analysis more difficult due to the vantage point of the trace (for instance, determining the initial congestion window is more difficult). In addition, so many variables are at play in such traces that direct, meaningful comparisons of the traffic are difficult. However, this sort of measurement study often produces many useful results (e.g., on the sizes of web transfers [TMW97]).
- We could have conducted an active measurement study by tracing web connections from our lab to various WWW servers. Such a study shares some of the problems with the study presented in this paper. For instance, all transfers share a portion of the network path, as well as a web client. This type of study is quite useful for taking certain measurements across a wide range of web servers (e.g., checking for web server conformance with the HTTP standard [KA99]).
- Finally, we could have used a mesh of hosts (such as NIMI [PMAM98, PA00]) to make active measurements. This mitigates some of the problems with taking measurements from a single client machine or a single server. However, it is not clear that such a mesh of hosts captures the true connectivity of a wide range of Internet web clients. Nor does such a mesh enable the measurement of a range of client network stacks used in the majority of traffic on the Internet.

While we may have been able to adapt one of the above approaches to our needs we feel that the strengths of our approach are ample and provided a rich variety of data about web clients that would not necessarily be available with other methodologies. The approach we chose does have several benefits. For instance, we were able to tweak the configuration of the web server and measure the effects (e.g., using a larger initial congestion window, as outlined in section 8). Also, our approach allows for the measurement of properties of a large number of real clients (e.g., which TCP options are supported in their stacks). While our approach is not without flaw, we believe (as outlined in [AF99]) there is no perfect way to assess Internet behavior and therefore believe that this survey can provide valuable insight into the performance of the web as seen from the user's perspective (since all traffic studied comes from web browsing as it happens "in the wild"). Throughout this

*This paper appears in ACM Computer Communication Review, 30(5), October 2000.

study we have attempted to measure only properties which are not subject to large biases due to the use of observations from only a single web server and note any biases we believe to be in the data presented. An item for future work will be to take such data from a number of web servers to gain a richer understanding of more attributes of client behavior.

The rest of this paper is organized as follows. Section 2 outlines our data collection techniques and discusses some preliminary analysis of the data. Section 3 outlines our measurements into how web clients utilize TCP connections to transfer web objects. Section 4 discusses the deployment status of various TCP options, as found in our data. Section 5 reports the round-trip times found in our data. Section 6 discusses the distribution of packet sizes found in our datasets. Section 7 reports our observations about the advertised window used by web clients and the possible impacts these window sizes have on performance. Section 8 gives an analysis of our web server's use of larger initial congestion windows. Finally, section 9 gives our conclusions and some future work in this area.

2 Preliminary Analysis

2.1 Data Collection

The data presented in this paper was collected between November 6, 1998 and March 24, 2000. For the large majority of our data collection period the server ran the NetBSD 1.3 operating system. On February 14, 2000, however, the operating system was upgraded to NetBSD 1.4. Therefore, roughly the last month of data presented used a slightly different network stack. We do not believe this influences the results presented in this paper because the TCP implementation was not drastically changed between the two versions. The web server used during the entire data collection period was Apache 1.2.6.

We used two main sources of information for the data presented in this paper. The first set of data used, denoted $\mathcal{L}$, consists of the Apache generated logs of each request made to the server. The second source of data is packet-level traces of the web traffic to and from the server, denoted $\mathcal{P}$. The packet-level traces were taken with *tcpdump* [JLM89] on the web server itself. We captured the first 100 bytes of 610,146,959 packets, with *tcpdump* reporting another 1,799 packets, or roughly 0.003%, dropped by the kernel[1]. We consider this an acceptably low amount of kernel packet loss that we did not attempt to correlate the kernel packet losses with specific connections in the trace file, as any effects of kernel drops should have a very negligible, if any, effect on our results.

We begin our investigation by examining the overall traffic pattern of the web server and removing data from our analysis for various reasons. The traffic patterns are unique to the particular web server we observed and therefore we cannot make any general claims from the patterns observed. This section is provided as an explanation of the datasets used and as background for the analysis outlined in the remaining sections.

Most of the analysis in this section is done in terms of web server hits. The number of hits is straightforward to obtain from the $\mathcal{L}$ dataset. However, without capturing the full packet contents (which we did not) and doing a good bit of analysis we cannot directly get the number of hits from the $\mathcal{P}$ dataset. So, for the $\mathcal{P}$ dataset we report the number of TCP connections traced.

Table 1 shows the number of server hits (or connections) as reported by each data set. For the purposes of this paper we are only interested in the web hits that traverse the wide-area Internet.

Category	$\mathcal{L}$	$\mathcal{P}$
Total Hits/Cnns	767,589	751,542
Local Hits/Cnns	20,172	45,516
WAN Hits/Cnns	747,417	706,026
Zero Length	21,536	23,189
Valid Hits/Cnns	725,881	682,837

Table 1: Web server Hits

Therefore, before the data analysis is conducted we remove all hits received by the server from hosts on the GRC network. As shown in the table, the number of local hits is relatively small. Note the number of local hits is greater in the $\mathcal{P}$ dataset when compared to the $\mathcal{L}$ dataset due to the lenient filter used on *tcpdump*. The filter captured all traffic to or from port 80. Therefore, roughly 25,000 random web transfers that occurred on the web server's network but had nothing to do with the web server in question were captured. We additionally remove any connections that did not transfer at least 1 byte of data in each direction. Such connections indicate some sort of failure in the client machine, the server host or the network between the two endpoints. While it is important to note that such connections do exist, we will not analyze them further. Therefore, unless otherwise indicated, the remainder of this paper includes analysis of only the valid wide-area hits. Finally, we note that the valid hits came from 50,194 distinct IP addresses. We expect this indicates that the hits cross a wide-variety of network paths, even though all connections share a portion of the path to the web server.

In several of the following sections the analysis could be significantly altered if a single host (or small group of hosts) were involved in a large portion of the connections we examine. Therefore, we created a second dataset from the $\mathcal{P}$ dataset by removing any connection to a host that was involved in over 1% of the connections in the $\mathcal{P}$ dataset. We denote this second dataset $\mathcal{P}'$. We removed 185,005 connections involving 11 IP addresses to obtain the $\mathcal{P}'$ dataset consisting of 497,832 TCP connections.

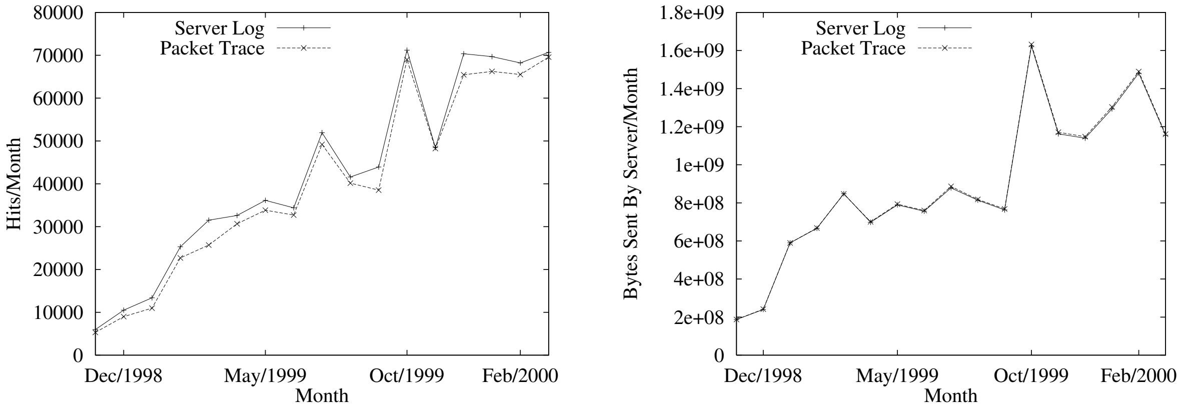
We used a slightly modified version of *tcptrace* 5.2.2b [Ost97] to analyze the $\mathcal{P}$ (and $\mathcal{P}'$) dataset. The changes made to *tcptrace* were to make the output easier to analyze or to make the tool report a particular piece of information that the standard version of the program does not report. We will note any additions made to *tcptrace* to do the analysis contained in the remainder of the paper. The output from *tcptrace*, as well as the data from set $\mathcal{L}$ is further analyzed using several short Perl and Bourne shell scripts.

2.2 Overall Traffic Patterns

Figure 1 illustrates the server activity as a function of time. The number of hits reported from the $\mathcal{L}$ dataset is greater than the number of TCP connections in the $\mathcal{P}$ dataset. This is explored in greater depth in section 3. As shown, the datasets are nearly identical in the number of bytes transmitted by the server. Both plots show an increase in traffic over the observation period. This is likely due to periodic addition of content to the server.

The server's transfer sizes are dictated by the content available on the particular server we observed. For instance, the median response size observed on our server is roughly two-thirds the size of the median response size reported in [Mah97], while the mean response size in our dataset is more than twice the value reported by Mah's data. This illustrates that response sizes could be much different if another server was observed. However, they provide some context for the results in the following sections. Figure 2 shows the mean and median transfer sizes over time. As illustrated, the mean transfer size is an order of magnitude greater than the median size.

[1] One of our trace files, consisting of 3,578,781 packets, failed to report the number of kernel drops (probably due to *tcpdump* being shutdown improperly). However, based on the likelihood of kernel drops in the other traces, we do not believe this presents a significant problem for the data analysis.



(a) Server hits per month.

(b) Bytes sent by the web server each month.

Figure 1: Web server activity over time.

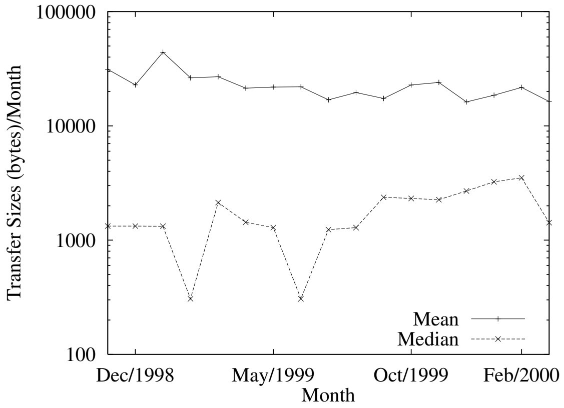


Figure 2: Web transfer sizes over time.

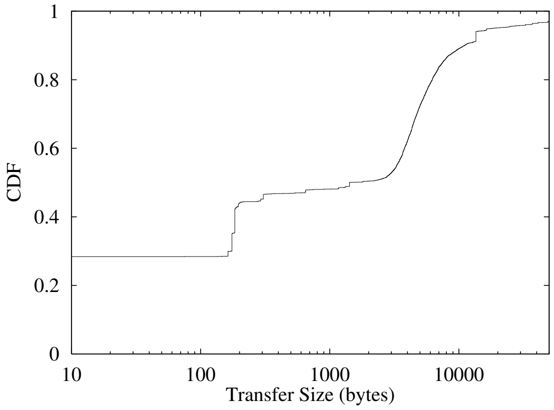


Figure 3: Distribution of transfer sizes for March 2000.

Meanwhile, the median transfer size is on the order of 1–5 packets. Figure 3 shows the distribution of transfer sizes for the last month of the dataset. As shown, over 90% of the transfer sizes are less than the mean transfer size reported in the previous figure. Also note that nearly 30% of the transfers are between 100–200 bytes long. These transfers mostly consist of HTTP headers and short HTML pages that indicate errors (file not found, forbidden, etc.).

While we do not delve into the reasons behind some of the spikes and dips in the above plots because we expect that such phenomena are a property of this particular server and its content, rather than based on some general network behavior. However, future studies should consider a more diverse set of web server data to verify this assumption.

3 HTTP Connection Usage

HTTP utilizes TCP connections in several different ways. Some HTTP browsers use *parallel* TCP connections to transfer the various objects that make up a web page (HTML code, graphics, etc.). Using this method the browser opens several connections at the same time and requests different objects on each connection. Another method that is supported in the HTTP protocol is for a client and server to use *persistent* TCP connections [Mog95]. Using this method, a TCP connection can be used to transfer multiple web objects. In this section we attempt to quantify the degree to which parallel and persistent connections are being utilized to transfer data from our web server. Our web server supports HTTP/1.1 persistent connections, as well as pipelining. However, this does not necessarily mean that web clients will request these features.

The use of HTTP connections can have performance, congestion control and resource utilization implications. For instance, using persistent connections with the pipelining option has been shown to improve web transfer speeds over satellite channels [KAGT00]. Meanwhile, using parallel connections can have a negative impact on end-to-end congestion control [BPS+98, FF99]. Specifically, a single loss causes one TCP connection to reduce *cwnd* by half. However, a single loss within a group of N connections causes the aggregate *cwnd* to by reduced by $1/2N$ yielding a more aggressive congestion control response. Finally, busy web servers have to manage system resources effectively. For instance, a web server may not want to keep an idle, persistent connection around as the connection requires memory and may slow control block lookups.

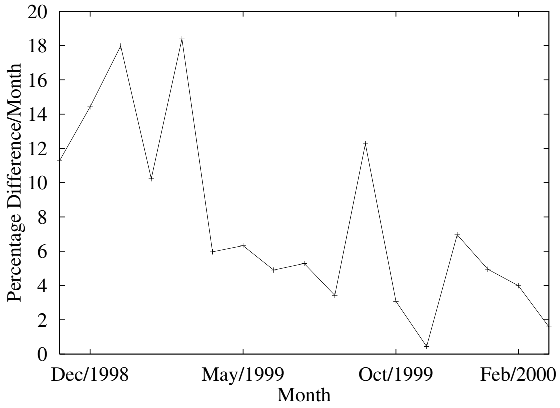


Figure 4: Percent difference between $\mathcal{L}$ reported hits and number of TCP connections in $\mathcal{P}$ dataset.

Figure 4 shows the percent difference between the number of hits reported in the $\mathcal{L}$ dataset and the number of TCP connections found in the $\mathcal{P}$ dataset. The figure indicates that persistent connections are being used to transfer multiple web objects on the same TCP connection. While noisy the figure seems to indicate that the use of persistent connections is reducing over time. However, additional data from a diverse set of web servers is needed to make a stronger conclusion.

Next we discuss the use of parallel HTTP connections. We define the *degree of parallelism* (DOP) as the maximum number of TCP connections open at the same time between the web server and a particular client over the course of our observation period. Our server communicated with 50,194 clients (IP addresses) during our observation period. Of these, 27,954 clients ($\approx 56\%$) made only a single connection with our server. These have been removed from further analysis in this section as there is no chance for the client to use parallel TCP connections.

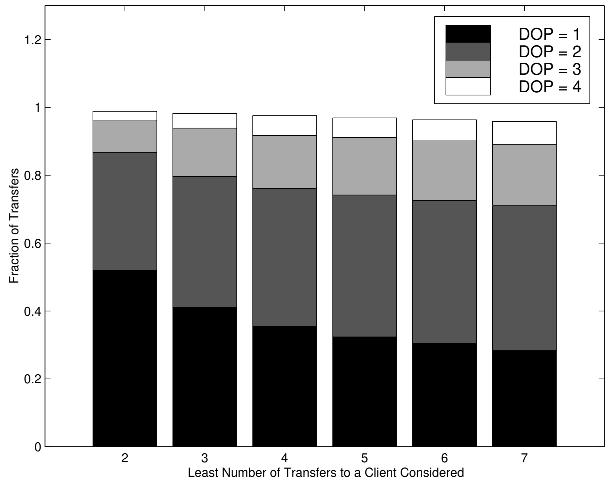


Figure 5: Use of parallel HTTP transfers.

Figure 5 shows the use of parallel TCP connections as a function of the least number of transfers between the server and a particular client considered. In other words, the far right-hand bar on the chart includes only data from clients that took part in at least 7 TCP connections in the $\mathcal{P}'$ dataset. The y-axis shows the percentage of clients using a particular DOP. The first thing to note is that nearly all connections used a DOP of 4 or fewer TCP connections (although, we observed DOPs as high as 25 connections[2]). As illustrated in the figure, the DOP increases as the number of transfers to a given client increases. This may indicate that clients making a small number of connections to the server (e.g., 2–3 connections) may do so at wide intervals and thus cannot make use of parallel connections. However, as clients transfer more objects the likelihood of using multiple parallel connections increases, and hence we note an increase in the percentage of connections using a DOP of more than 1 connection. From figure 5 we can see that approximately two-thirds of web clients use parallel TCP connections to download web pages, with the most popular DOP being 2 connections.

We found that quantifying the use of persistent and parallel HTTP connections was difficult with our datasets. The analysis above is tentative and all of our questions could not be answered conclusively by our data. We are currently modifying our web server's logging routines to include more information, such that future analysis will be more straightforward and more accurate. One particular addition to the logs will be a unique *connection identifier* (CID) that will be logged with each object request. The CID will allow for correlation of exactly which connections were persistent and which were utilized in parallel.

4 Use of TCP Options

As TCP has evolved, several options have been added to the protocol to make it perform better in certain environments. The goal of the analysis presented in this section is to assess the deployment status of various TCP options. This serves two purposes. First, it gives network engineers a good idea about the features they may be able to expect from end hosts in the Internet. In addition, this analysis sheds light on what options researchers may want to simulate when investigating TCP.

Note that the discussion of TCP's maximum segment size (MSS) option is deferred until section 6.

The features supported on a particular connection in our dataset are likely to be largely determined by what operating system the user is running. [Mah99] provides a list of which features are supported in current operating systems. Therefore, the data presented in this paper may also indicate the proportion of hosts using various operating systems or how up-to-date user's OS version is kept. While interesting, we do not delve into this topic further in this paper.

In this section we analyze several TCP options by looking at the percentage of connections and bytes transmitted by the server to clients that support the given option. The number of bytes transmitted using a given option may be somewhat biased by the size of content provided by our web server. Likewise, the percentage of connections may also be biased by the number of hits required to load the web pages on our server (which could be different from the make-up of pages on different servers). In addition, if a large number of connections come from a relatively small number of clients, the $\mathcal{P}$ dataset could be biased. Hence, we also measure the the percentage of hosts using the given option which should not be biased by our particular web server. Our results could be biased if the

[2]The web server logs indicate that the use of a large number of parallel TCP connection is usually caused by a client harvesting a large number of e-mail messages from our web archives of various mailing lists.

sample of clients found in our dataset is not representative. However, our dataset is large enough that we do not believe this to be the case.

4.1 High Performance Options

We first focus our attention on the options added to TCP by RFC 1323 [JBB92] for high performance over network paths with large amounts of bandwidth and/or long delays. RFC 1323 added the window scaling and timestamp options. The window scaling option is used during the three-way handshake that starts each TCP connection. Each host announces a scale factor. The sending host right shifts the desired advertised window by the scale factor before transmitting the advertised window. The receiving host will then left shift the advertised window in all incoming packets by the scale factor before using the advertised window size. This allows TCP connections to utilize an advertised window of more than the 64 KB provided by the original TCP specification [Pos81], which is required for operation over long, high-bandwidth networks [JBB92].

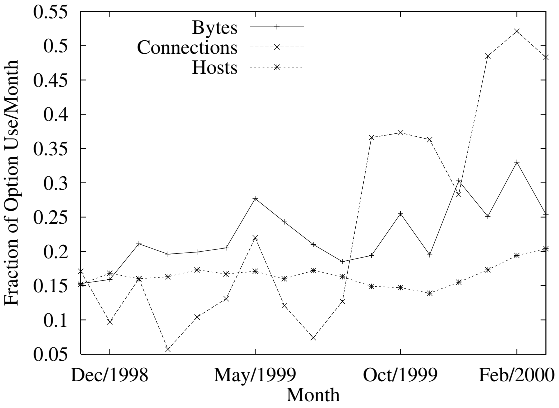


Figure 6: Use of window scaling option over time.

To gain an idea about the deployment of the window scaling option we analyzed the $\mathcal{P}$ dataset to determine prevalence of the option. Figure 6 shows the use of the window scaling option over the measurement period. As shown in the plot, the percentage of hosts supporting the window scaling option was fairly stable over the measurement period at 15-20%. The percentage of connections using window scaling varies widely over the survey. However, during the last three months of the survey the number of connections supporting window scaling rose sharply to roughly 50%. Since we do not see similar percentages for the number of hosts supporting window scaling the plot indicates that a relatively small number of hosts have likely upgraded to support window scaling and are responsible for a disproportionate number of connections. We verified this by analyzing the trace files and noting that the majority of the connections using window scaling came from two clients (IP addresses) that recently started using window scaling. Finally, the number of bytes transmitted by the server using the window scaling option varied somewhat during the measurement period.

Next, we analyzed the scale factor advertised by the web clients. We found that just over 84% of the clients advertised a scale factor of zero. This indicates that they are willing to scale their peer's (the server's) advertised window, but would not be scaling their own advertised window. Nearly all of the remaining hosts advertised a scale factor of one, however, we observed scale factors as high as 12 (in two hosts). Note the implications of the advertised window sizes found in the $\mathcal{P}$ dataset are explored further in section 7.

RFC 1323 also introduces the timestamp option to be used in conjunction with window scaling. Since a TCP with the window scaling option can cycle through the sequence space provided by TCP much faster than when the advertised window is limited to 64 KB, TCP needs additional protection against passing old data to the application. The timestamp option calls for the sender to insert a timestamp in each packet that is transmitted. In addition, the most recent timestamp received from the remote host is also echoed. The timestamp option has two purposes. First, timestamps are used in conjunction with window scaling in the *Protect Against Wrapped Sequences* (PAWS) algorithm. Second, timestamps are used to obtain better and more frequent RTT measurements (although there is recent evidence that this use of timestamps is not particularly useful [AP99]).

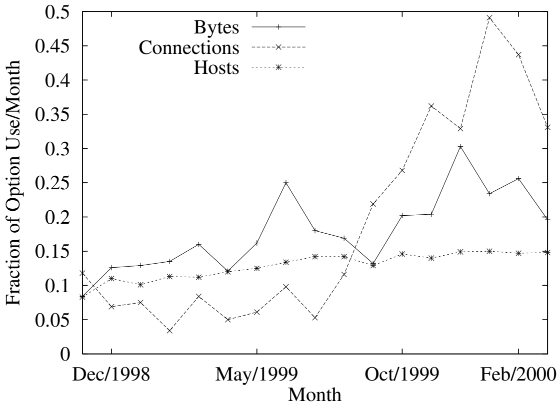


Figure 7: Use of timestamp option over time.

Figure 7 shows the prevalence of the timestamp option in web clients. The history of timestamp option use is similar to that of the window scale option. The percentage of connections using timestamps varies widely and becomes quite large towards the end of our dataset. One of the two hosts that made the majority of the connections to the server and started to support window scaling in recent months (as discussed above) also started using timestamps. This explains the increase in connections supporting the timestamp option towards the end of the dataset. As with the window scaling option, it appears as though the percentage of hosts using timestamps is roughly stable (or slowly increasing) throughout the observation period.

Finally, we note that in total, 11% of the web clients observed in our survey used both timestamps and window scaling, while another 2.8% used only timestamps and 5.7% used only window scaling.

4.2 Selective Acknowledgments

Next we focus on the selective acknowledgment (SACK) option defined in RFC 2018 [MMFR96]. SACKs are used to improve upon TCP's original method of informing the sender about which segments have arrived at the receiver. As defined in [Pos81], TCP uses a cumulative acknowledgment that informs the sender of the last in-order byte of data that has arrived at the receiver. Using the SACK option, the receiver can inform the sender about arbitrary segments that have been received, regardless of the order in which they arrived. This allows the sender's TCP to employ more advanced loss recovery and congestion control algorithms [FF96].

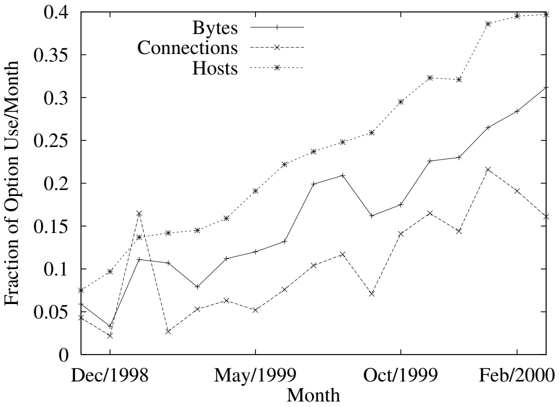


Figure 8: Use of SACK option over time.

Figure 8 shows the prevalence of the SACK option in the $\mathcal{P}$ dataset. Note that our web server does not support SACK. Therefore, the percentages reported are the number of connections (hosts, bytes) that would have used selective acknowledgments had the server supported them. In other words, the number of clients that advertised "SACK permitted" in the three-way handshake. As shown, the number of clients supporting SACK is steadily growing from roughly 8% at the end of 1998 to nearly 40% by March, 2000. The number of connections and bytes utilizing SACK is lagging behind the percentage of hosts supporting the option. This indicates that a number of the web crawlers that hit our server many times per month do not support the SACK option yet. We believe the SACK deployment shown in this plot is consistent with the recommendation made in [AF99] that SACK should be a part of all TCP investigations, as SACK is clearly steadily being deployed in the Internet.

5 Round-Trip Times

This section focuses on examining the distribution of round-trip times (RTT) between the server and the clients. We used *tcptrace* to produce the average and median RTT for each connection in the $\mathcal{P}$ dataset. The tool takes an RTT sample for each non-retransmitted segment and the corresponding ACK. Our purpose in investigating the distribution of RTTs is twofold. First, such data provides researchers with realistic RTTs to build into their simulations. As suggested in [PF97, AF99] a range of parameters should be used in simulating networks. The data presented in this section provides some guidance on what a reasonable range of RTTs might look like. The second goal is to assess the degree that saving RTTs in transfers is important (i.e., if the RTT is negligible and cannot be detected by a user maybe we do not need to spend time trying to squeeze every possible extraneous RTT out of transport and application layer protocols).

There are two instances of bias that may be introduced into our measurements. First, all transfers between the server and the remote clients share a portion of the network path. Therefore, if that portion of the path is congested or imposes a large delay the measurements will all be biased by the location of the server. We note that we observed very few (less than 1%) RTT samples under 15 ms in the $\mathcal{P}$ dataset. In the $\mathcal{P}'$ dataset the minimum RTT appears to be approximately 40 ms. This indicates the the location of the server generally imposes a modest minimum RTT on the samples obtained. (Note: This may be true of *any* server, but more data is required to make such a claim.) Since the distribution of RTTs is not concentrated around the minimum RTT for a given dataset we believe that the shared portion of the network path is not seriously biasing our measurements.

The second form of bias is from *tcptrace*'s use of Karn's algorithm [KP87] to take the RTT measurements. If a particular segment was needlessly retransmitted we do not observe the RTT associated with the original data packet and its corresponding acknowledgment. [AP99] shows that the standard BSD retransmission timeout (RTO) mechanism rarely causes needless retransmissions. However, [BPS99] shows that reordering is not necessarily a rare occurrence in the Internet. Reordering can cause TCP to retransmit segments prematurely via the fast retransmit algorithm [Jac88]. However, we do not believe that such an event necessarily triggers a change in RTT that *tcptrace* misses by ignoring the ambiguous RTT sample.

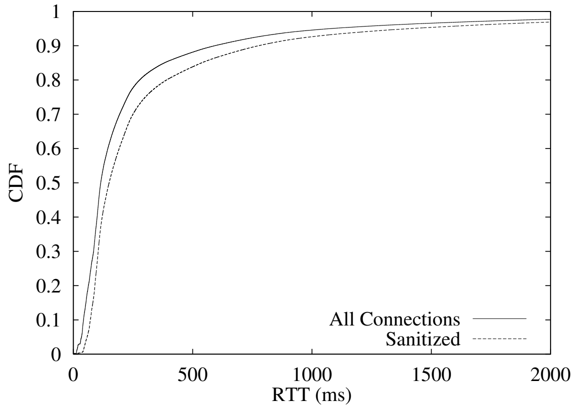


Figure 9: Distribution of average RTTs.

Figure 9 shows the distribution of average RTTs for each connection in the $\mathcal{P}$ and $\mathcal{P}'$ datasets. Note that the RTT reported is composed of not only the time required for the data packet and corresponding ACK to traverse the network path, but also processing time at the receiver. For instance, a client using delayed ACKs [Bra89, APS99] may refrain from transmitting an ACK for up to 500 ms, which would inflate the RTT. The delayed ACK mechanism could, therefore, skew the average RTT reported. Many implementations use a 200 ms heartbeat timer to trigger delayed ACKs. This causes a 100 ms delay on average when transmitting timer-based ACKs. We expect this effect to be small as the transfer size increases and we get a larger number of RTT samples. However, most of our transfers are short and therefore the delayed ACK timer may be skewing our data a bit, however quantifying the skew is difficult with out dataset.

While both distributions shown in the figure have the same basic shape we note that the connections in the $\mathcal{P}'$ dataset have longer RTTs than when considering all the connections in $\mathcal{P}$. This indicates that the connections removed from $\mathcal{P}$ to yield the $\mathcal{P}'$ dataset were skewing the distribution towards smaller RTTs. The host names of the IP addresses not included in the $\mathcal{P}'$ dataset indicate that the clients are web crawlers, surveying the content on our server for search engines. We expect such clients to enjoy good connectivity to the Internet, explaining why they have generally lower RTTs than the rest of the clients. As indicated in the figure, approximately 85% of the RTTs are between 15–500 ms. This gives a nice range of RTT values for researchers constructing internetworking simulations.

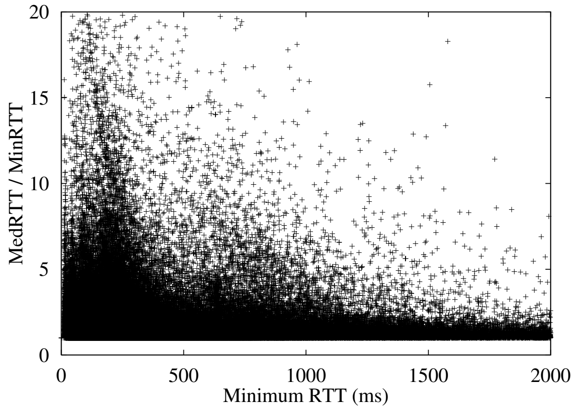


Figure 10: Comparison of the minimum and median RTTs a connection observes.

Figure 10 provides a comparison of the minimum RTT observed and the median RTT for each connection. The *x*-axis is the minimum RTT in milliseconds, while the *y*-axis is the median RTT for the same connection as a multiple of the minimum RTT. The data from the $\mathcal{P}'$ dataset is shown in this plot. To highlight the behavior of the vast majority of the connections the *x*-axis is limited to 2 second minimum RTTs, as in the last plot. While our dataset shows median RTTs as high as 200 times the observed minimum RTT we limited the *y*-axis to a factor of 20 to better illustrate the behavior of the vast majority of the connections in the dataset. The median RTT was within a factor of 2 of the minimum RTT in slightly over 90% of the connections when considering all connections in the $\mathcal{P}'$ dataset. However, the plot illustrates that for shorter RTTs the variability within connections is sometimes quite large. (We found one connection with a median RTT of 200 times the minimum RTT!) One explanation for this decrease in variability as the RTT grows is the use of a network link with a high delay (e.g., a satellite channel) that has the effect of drowning out the variability in the rest of the network path. However, this cannot be further investigated without additional data.

Another note about this data is that the minimum RTT may come from a short segment (e.g., a SYN). On slow links the transmission time of a short packet can be significantly shorter than that of a full-sized data segment, which could explain some of the variability shown in the figure. However, most TCP implementations we are aware of do not take packet size into account when measuring RTTs. Therefore, we believe this figure presents an accurate view of the network from the perspective of a TCP data sender.

We also note that as shown in section 2 the majority of the transfers from our web server are very short. Together with figure 10 this indicates that RTTs can change significantly on short time scales over some network paths. A possible area of future work is to assess the stationarity of RTTs in the network (much as has been done for routes, loss rate and throughput [ZPS00]).

We now turn our attention to the second goal of this section. RFC 1144 [Jac90] suggests 100–200 ms as the amount of time that users can perceive in regards to responses from networks. We note that figure 9 shows that nearly 75% of the connections in the $\mathcal{P}'$ dataset experience average RTT delays over 100 ms and nearly 40% of the RTTs observed exceed 200 ms. This indicates that paying careful attention to making transport and application protocols use fewer RTTs (when possible) is important. For instance, proposals such as *limited transmit* which allows TCP to transmit new data segments on the first two duplicate ACKs to save retransmission timeouts [ABF00] can be important changes. As indicated by the data, saving even several RTTs may represent a significant improvement for users.

An additional note about the RTT distribution is that slightly over 2% of the connections in our dataset observed at least one RTT over 3 seconds. Furthermore, slightly more than 1% of the connections averaged RTTs of over 3 seconds. This indicates that TCP's minimum initial retransmission timeout (RTO) of 3 seconds as specified in [Bra89, PA00] continues to be a conservative choice.

6 Packet Sizes

Next we analyze the packet sizes used by the server when transferring data to the clients in our dataset. We will use this analysis to draw conclusions in the next two sections. In addition, understanding the packet sizes used in real networks will enable researchers to simulate wisely.

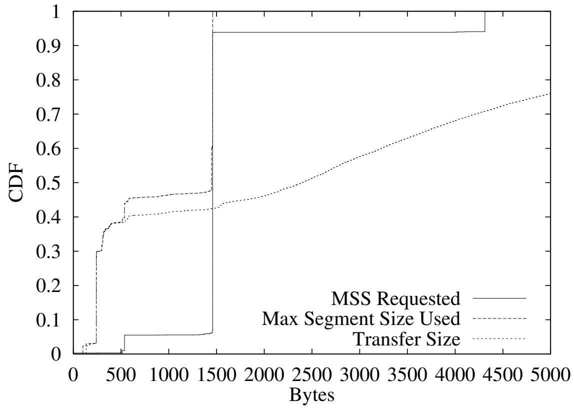


Figure 11: Distribution of packet sizes.

Figure 11 shows the distributions of the maximum segment size (MSS) requested by the clients in the SYN exchange, the largest packet size used by a connection and the transfer size for the $\mathcal{P}$ dataset. As shown, nearly 90% of connections advertised an MSS of roughly 1460 bytes in the SYN segment. Roughly 5% of the connections advertised a lower MSS (around 500 bytes). Approximately 6% of the transfers advertised maximum segment sizes of around 4000 bytes. We found 27 connections that advertised an MSS over 17,000 bytes (although, 15 connections were to a single client host). We used SNMP [CFSD88] to query the last hop router to the clients advertising an MSS of over 17,000 bytes. The one router that answered our query supported an MTU of 4,180 bytes on two of its six interfaces and an MTU of 1,500 bytes on the rest. This does not explain the large advertised MSS. This suggests either a bug in the TCP stack causing a large MSS to be advertised or that the network has changed between the time the connection was made and our SNMP query.

The figure shows that, as expected, the maximum packet size and the total transfer size track quite well when the transfer size is less than 500 bytes. When transfer sizes exceed roughly 500 bytes the transfer size no longer tracks the maximum packet size. We see that roughly 5% of the transfers use a maximum packet size of approximately 500 bytes, as expected from the MSS advertisements. The remaining transfers use a maximum segment size of roughly 1460 bytes (also, as predicted by the MSS options observed). We note that no transfers use packet sizes greater than

1500 bytes because the server is connected via a 10 Mbps Ethernet with a 1500 byte MTU and hence does not send larger packets.

We conclude that in our sample 1500 byte packets are used the vast majority of the time (when the transfer size is large enough to support their use). While this would be a stronger result if we had datasets from additional servers we believe researchers are fairly safe using 1500 byte packets in simulations and emulations.

7 Advertised Windows

This section focuses on the advertised window size used by web clients. The advertised window represents the data receiver's upper bound on the amount of outstanding data, or data that has been transmitted but for which an acknowledgment has not yet arrived. Therefore, the advertised window can have a direct impact on the performance of a data transfer, as outlined in [SMM98, AF99].

In addition to the advertised window, the *congestion window* (*cwnd*) is a sender-side state variable that represents the actual amount of outstanding data the sender is permitted to inject into the network. The value of *cwnd* is limited by the advertised window. TCP uses the *slow start* algorithm [Jac88, APS99] to increase the value of *cwnd* at the beginning of a transfer. The algorithm starts by setting *cwnd* to 1 segment and then sending 1 segment (or, sometimes a small number of segments, see the next section) and waiting for the corresponding acknowledgment (ACK). For each ACK received during slow start *cwnd* is increased by 1 segment. The algorithm ends when congestion is detected (either inferred from observing packet drops or from Explicit Congestion Notification (ECN) [Flo94, RF99]) or when *cwnd* reaches the advertised window.

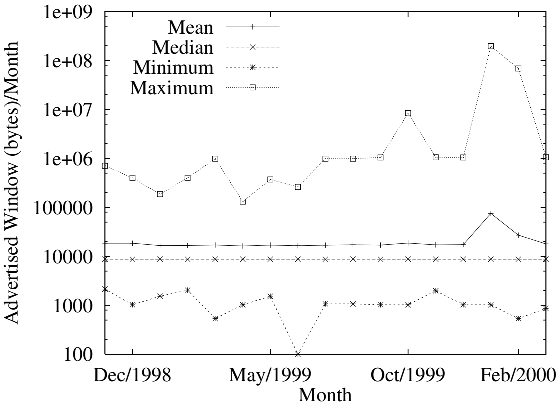


Figure 12: Advertised windows used by web clients over time.

Figure 12 shows the advertised window sizes from the $\mathcal{P}$ dataset over time. The advertised windows represent the maximum advertised window during the connection, as reported by *tcptrace*. As shown, the mean and median advertised window size has remained approximately the same over the course of our observation. The median transfer size is 8,760 bytes, or 6 packets if we use 1,460 byte segments, as suggested by the data presented in the last section. Meanwhile, the average advertised window is roughly 18 KB, or approximately 12 packets. The mean advertised window jumps noticeably in January and February, 2000. These two spikes are caused by 3 connections which use a very large advertised window (as shown by the maximum advertised window line). When these 3 connections are removed from the analysis, the two months in question are no longer distinguishable from the other months of the study, in terms of average advertised window. Note that an analysis using the $\mathcal{P}'$ dataset yields nearly identical results.

TCP uses the fast retransmit algorithm to quickly detect packet loss [APS99, Jac88]. TCP receivers will send *duplicate acknowledgments* in response to segments arriving out-of-order. TCP senders use the receipt of 3 duplicate ACKs as an indication that a given segment has been lost. The segment is retransmitted and *cwnd* is halved because the drop is assumed to indicate network congestion. Therefore, a *cwnd* of less than 4 segments prevents TCP's fast retransmit algorithm from being triggered. For instance if *cwnd* is 3 segments and one segment is dropped by the network the sender will receive only two duplicate ACKs (assuming no ACK loss) and will then wait for the retransmission timer to expire to resend the dropped packet.

Morris [Mor00] extends to above argument further. In order to stay out of the regime where TCP frequently uses the RTO to recover from loss the minimum *cwnd* should be 4 segments. In order to always have 4 segments in the network the *cwnd* needs to be able to grow to at least 8 segments, such that when congestion is detected and *cwnd* is halved, *cwnd* is still at least 4 segments. From the data we have collected it seems that the advertised window will likely prevent the *cwnd* from reaching 8 segments in the majority of the cases. This argues that default advertised windows should be increased.

Rather than increasing the advertised window size, several researchers have suggested that TCP send new segments upon the reception of the first two duplicate ACKs [BPS$^+$98, LK98, ABF00]. This will trigger additional duplicate ACKs (if appropriate) and therefore fast retransmit will be invoked.

However, the above algorithm does not aid short connections that have no new data to transmit in response to duplicate ACKs. ECN [Flo94, RF99] provides a possible mitigation to this problem. Rather than dropping the segments, the network could simply mark them as experiencing congestion. This would allow the connection to quickly complete without requiring a costly retransmission timeout. Another possibility is for TCP to detect that (i) one duplicate ACK has arrived, (ii) there is no more data to send and (iii) based on the number of outstanding segments 3 duplicate ACKs cannot be expected. In this case, the TCP would trigger fast retransmit on a smaller number of duplicate ACKs.

Semke [SMM98] argues for the use of an automatic socket buffer tuning algorithm and the notion that the network should dictate the performance of a TCP connection, rather than being limited by some arbitrary limit placed on the transfer by one of the endpoints (i.e., the advertised window). (Note that the advertised window is not always arbitrary, but we believe it often has little to do with the current network or host conditions).

As noted above, the advertised window can limit a connection's performance. The maximum throughput a TCP can obtain is given by equation 1 [Pos81], where T is the throughput, W is the advertised window size and RTT is the round-trip time between the sender and the receiver.

$$T = \frac{W}{RTT} \tag{1}$$

As indicated by the equation, if W is too small, such that T is less than the available bandwidth, the connection will not be able to utilize the available resources of the network. Using automatic buffer tuning [SMM98] effectively removes the advertised window limitation in all cases except when the end host needs to limit the buffer size due to memory constraints.

Assuming no congestion indications, TCP is required to send at least $2 \cdot W$ bytes of data to open and fill a congestion window equal to an advertised window of W bytes. Connections whose congestion window reaches the advertised window are likely limited by the end systems in the connection, rather than by the capacity of the network. We are interested in assessing how often this happens.

Many of the transfers in the $\mathcal{P}$ dataset are not long enough for the advertised window to become a factor. This is likely caused by the content our web server provides. However, we note that [TMW97] shows typical web transfers are between 9–12 KB, which is also too small to fill and utilize the median advertised window sizes. We found that 644,102 connections from the $\mathcal{P}$ dataset failed to send enough data to become limited by the advertised window. Of the remaining 38,735 connections, we found that 27,066 connections, or nearly 70%, were limited by the advertised window size. This indicates that in the cases where TCP would likely have been able to obtain better performance the advertised window size hindered the throughput obtained. We believe this result provides further evidence that default advertised window sizes should be increased, or automatic buffer tuning [SMM98] should be employed.

In related work, [BPS$^+$98] reports that 14% of connections to a busy web server are limited by the client's advertised window size. In our sample, only 4% of the total number of connections are limited by the advertised window. However, looking at the total percentage of connections without regard to transfer size can distort the results.

8 Larger Initial Congestion Window

The current TCP congestion control specification [APS99] allows TCP implementations to use an initial congestion window of up to 2 segments. The *cwnd* is increased from this initial value using the slow start algorithm. RFC 2414 [AFP98], an experimental document within the IETF, proposes allowing TCP to use initial *cwnd* values of up to 4 segments, depending on the segment size. Specifically, [AFP98] proposes using equation 2 to set the initial *cwnd* size.

$$cwnd = min(4 \cdot MSS, max(2 \cdot MSS, 4380\ bytes)) \qquad (2)$$

NetBSD 1.3 implements larger initial congestion windows as given in equation 2 as an option that can be enabled by the system administrator for experimentation. We enabled the option on our web server to investigate the impact of using a larger initial *cwnd* with realistic Internet traffic. We hope this provides some input to the IETF community if and when the proposal to use an initial *cwnd* of 3–4 segments moves onto the standards track.

NetBSD keeps *cwnd* in terms of bytes rather than segments. This distinction is important because while the spirit of equation 2 places an upper bound on the number of segments in the initial burst of data sent into the network, the NetBSD implementation does not necessarily do so. An example we found in our data quite often occurs when using an MSS of 1460 bytes. According to equation 2 this should yield an initial *cwnd* of 4380 bytes (or 3 full-sized segments). But, the server uses the `write()` system call to write 2 chunks consisting of 2048 bytes each. This yields 4 segments in the initial *cwnd* rather than 3 segments. The first and third segments are each 1460 bytes, while the second and fourth are 588 bytes. So, by not writing a large chunk of data initially, the server causes the initial burst of data into the network to be more segments than allowed, yet less bytes than allowed by equation 2.

In addition, NetBSD 1.3 contains a well-known TCP bug, whereby the ACK of the SYN-ACK in the three-way handshake causes *cwnd* to be incremented as if the connection were in slow start [All97, PAD$^+$99]. This allows for initial congestion windows of one segment more than allowed by equation 2. We found that in our dataset the size of the initial value of *cwnd* in bytes is always less than or equal to the value predicted by equation 2 plus one MSS, however the number of segments ranged between 1–7.

Assessing the performance impact of using a larger initial *cwnd* from our data is difficult. All connections had the opportunity to use a larger initial *cwnd* if enough data was transmitted on the particular connection. So, the connections using small initial *cwnd* values also transferred a small amount of data. This makes for a difficult comparison with connections using a larger initial *cwnd* (and hence transfered more data). We are currently taking a second set of data that only enables the larger initial *cwnd* option on some of the transfers. This should allow us to more easily compare the performance of connections using various initial *cwnd* values in future studies.

Initial *cwnd* (segments)	**% Using *cwnd***	**% With Loss In Initial *cwnd***
0	1.9/2.5	N/A
1	40.0/28.9	1/1
2	13.7/17.2	2/3
3	13.6/15.4	2/2
4	8.4/8.4	2/2
5	20.8/25.7	3/3
6	1.6/1.8	5/5
7	0.0/0.0	3/3

Table 2: Initial *cwnd* sizes and corresponding loss rates.

We can, however, analyze the amount of additional loss a larger initial burst of data creates in the first burst of traffic. Table 2 shows the percentage of connections in the dataset that used each initial *cwnd* size. The first percentage given is the percentage of all connections in the $\mathcal{P}$ dataset, while the second value is the percentage from the $\mathcal{P}'$ dataset (i.e., after removing the heaviest users of the web server, as detailed in the previous sections). The last column of the table shows the percentage of connections using the given initial *cwnd* value that retransmitted segments from the initial window of data[3] (again, percentages for $\mathcal{P}$ and $\mathcal{P}'$ are given). The first row of the table, reporting an initial *cwnd* of zero indicates the percentage of connections in which the SYN is lost. The distribution of initial *cwnd* values is determined by the segment sizes used (as discussed in section 6), as well as by the amount of data being sent on a particular TCP connection.

As shown in the table, we observed initial *cwnd* sizes of more than 4 segments due to the implementation issues discussed above. The table shows that 40% of the connections in $\mathcal{P}$ used an initial window of 1 segment. This agrees with the transfer size distribution shown in figure 11 in which roughly 40% of the transfers are less than 1500 bytes (or 1 segment in most cases). The table shows that using initial *cwnd* values of 2–4 segments, as suggested in [AFP98], slightly increases the percentage of connections that experience loss in the first window of data transmitted when compared to using a 1 segment initial *cwnd*. This is consistent with previous studies (e.g., [AHO98]).

Finally, table 2 shows that using a 3–4 segment initial window does not increase the chance of loss in the initial burst of data over that of using 2 segments. This is an indication that using 3 or 4 segment initial windows is safe for general use in the Internet. However, initial *cwnd* sizes of more than 4 segments seem to increase the initial loss rate more when compared to a standard initial window size. This indicates that using such initial *cwnd* sizes may be

[3] Note that *tcptrace* required modifications to report the number of retransmits in the initial window of data.

inappropriate in shared networks.

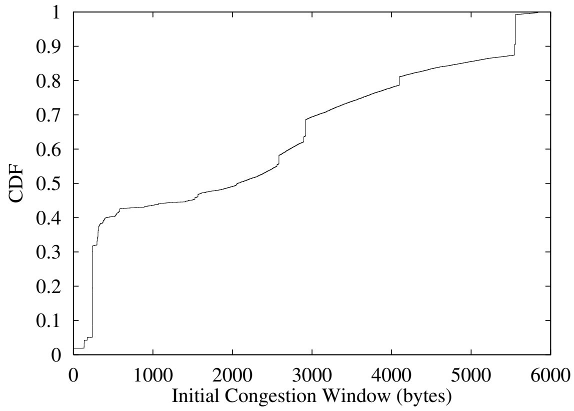


Figure 13: Distribution of initial *cwnd* in terms of bytes.

Figure 13 shows the distribution of the value of the initial *cwnd* in terms of bytes. We expected more pronounced plateaus in the plot, representing the initial *cwnd* obtained by using popular segment sizes with equation 2. As discussed above, there are two problems with this expectation. First, the sizes of the transfers often dictated the initial *cwnd* utilized. Second, the web server's use of the `write()` system call to cause short segments to be transmitted also influences the results.

We cannot say with certainty that Apache's writing of small chunks at the beginning of a transfer is common, given that we only observed a single server (even though Apache is a popular web server). However, we recommend that application developers send larger chunks of data to TCP, rather than writing data in such a way that causes TCP to send small segments when more data is immediately available. This is especially important for the first write of the transfer, which should be at least 4380 bytes (when possible) to handle larger initial congestion windows. Or, at a bare minimum roughly 3000 bytes to cover the initial congestion window allowed by RFC 2581, when using the popular 1500 byte segment size.

9 Conclusions and Future Work

The following are the key results and recommendations from the analysis performed for this paper.

- The SACK option is being steadily deployed in web client TCP stacks. Researchers conducting TCP simulations should include SACK based TCPs, as suggested in [AF99].
- Our data indicates that web client's advertised window sizes are currently too small, in general. The small advertised windows likely limit performance in roughly 70% of the transfers that are long enough to fully utilize the advertised window. In addition, small advertised windows may hinder loss recovery. This can be mitigated by increasing the advertised window size or improving TCP's loss recovery algorithms, as outlined in section 7 and [ABF00].
- Using larger initial congestion windows, as proposed in [AFP98], does not drastically increase the number of TCP connections that experience loss in the first burst of data, indicating that using larger initial values for *cwnd* is appropriate in most network paths.
- Approximately 85% of the average RTTs observed are between 15–500 ms, giving researchers a nice range of RTTs to use in simulations.
- WWW clients use of persistent connections seems to be declining, while the use of parallel TCP connections to transfer web objects has remained fairly stable. However, more data would be useful in clarifying these points.

In addition, this paper has suggested several items for future work, as follows.

- Collecting data from multiple web servers for analysis would provide stronger results and allow more general conclusions.
- A survey of many web servers for some of the same information would be useful, as well.
- Instrumenting web servers to better study client use of parallel and persistent connections would be useful.

Acknowledgments

Shawn Ostermann provided much needed assistance with his very nice *tcptrace* analysis tool. Ethan Blanton, Sally Floyd, Joseph Ishac and Vern Paxson provided detailed feedback and discussions on an early draft of this paper. Finally, the CCR reviewers provided a number of excellent suggestions on ways to improve this paper. My thanks for all!

References

[ABF00] Mark Allman, Hari Balakrishnan, and Sally Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit, August 2000. Internet-Draft draft-ietf-tsvwg-limited-xmit-00.txt (work in progress).

[AF99] Mark Allman and Aaron Falk. On the Effective Evaluation of TCP. *Computer Communication Review*, 29(5):59–70, October 1999.

[AFP98] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window, September 1998. RFC 2414.

[AHO98] Mark Allman, Chris Hayes, and Shawn Ostermann. An Evaluation of TCP with Larger Initial Windows. *Computer Communication Review*, 28(3), July 1998.

[All97] Mark Allman. Fixing Two BSD TCP Bugs. Technical Report CR-204151, NASA Lewis Research Center, October 1997.

[AP99] Mark Allman and Vern Paxson. On Estimating End-to-End Network Path Properties. In *ACM SIGCOMM*, September 1999.

[APS99] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP Congestion Control, April 1999. RFC 2581.

[BPS+98] Hari Balakrishnan, Venkata Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *IEEE InfoCom*, March 1998.

[BPS99] Jon Bennett, Craig Partridge, and Nicholas Shectman. Packet Reordering is Not Pathological Network Behavior. *IEEE/ACM Transactions on Networking*, December 1999.

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

[CFSD88] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol, August 1988. RFC 1067.

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

[FF99] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(6), August 1999.

[FGM+97] R. Fielding, Jim Gettys, Jeffrey C. Mogul, H. Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, January 1997. RFC 2068.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. *Computer Communications Review*, 24(5):10–23, October 1994.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

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

[JBB92] Van Jacobson, Robert Braden, and David Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[JLM89] Van Jacobson, Craig Leres, and Steven McCanne. *tcpdump*, June 1989. Available via anonymous FTP from ftp.ee.lbl.gov.

[KA99] Balachander Krishnamurthy and Martin Arlitt. PRO-COW: Protocol Compliance on the Web. Technical Report #990803-05-TM, AT&T Labs, August 1999.

[KAGT00] Hans Kruse, Mark Allman, Jim Griner, and Diepchi Tran. Experimentation and Modeling of HTTP Over Satellite Channels. *International Journal of Satellite Communication*, 2000. To appear.

[KP87] Phil Karn and Craig Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *ACM SIGCOMM*, pages 2–7, August 1987.

[LK98] Dong Lin and H.T. Kung. TCP Fast Recovery Strategies: Analysis and Improvements. In *Proceedings of InfoCom*, March 1998.

[Mah97] Bruce Mah. An Empirical Model of HTTP Network Traffic. In *Proceedings of INFOCOM 97*, April 1997.

[Mah99] Jamshid Mahdavi. Enabling High Performance Data Transfers on Hosts. Technical report, Pittsburgh Supercomputer Center, June 1999. http://www.psc.edu/networking/perf_tune.html.

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

[Mog95] Jeffrey C. Mogul. The Case for Persistent-Connection HTTP. In *ACM SIGCOMM*, pages 299–313, 1995.

[Mor00] Robert Morris. Scalable TCP Congestion Control. In *IEEE INFOCOM*, March 2000.

[Ost97] Shawn Ostermann. *tcptrace*, 1997. Available from http://jarok.cs.ohiou.edu/.

[PA00] Vern Paxson and Mark Allman. Computing TCP's Retransmission Timer, April 2000. Internet-Draft draft-paxson-tcp-rto-01.txt (work in progress).

[PAD+99] Vern Paxson, Mark Allman, Scott Dawson, William Fenner, Jim Griner, Ian Heavens, Kevin Lahey, Jeff Semke, and Bernie Volz. Known TCP Implementation Problems, March 1999. RFC 2525.

[PF97] Vern Paxson and Sally Floyd. Why We Don't Know How to Simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[PMAM98] Vern Paxson, Jamshid Mahdavi, Andrew Adams, and Matt Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications*, 1998.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

[RF99] K. K. Ramakrishnan and Sally Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP, January 1999. RFC 2481.

[SMM98] Jeff Semke, Jamshid Mahdavi, and Matt Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM*, September 1998.

[SP00] Jonathan Stone and Craig Partridge. When The CRC and TCP Checksum Disagree. In *ACM SIGCOMM*, September 2000.

[TMW97] Kevin Thompson, Gregory Miller, and Rick Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, November/December 1997.

[ZPS00] Yin Zhang, Vern Paxson, and Scott Shenker. The Stationarity of Internet Path Properties: Routing, Loss and Throughput. Technical report, ACIRI, May 2000.