NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-06-15
October 16, 2006

On the Use of Parallel Connections in Web Browsers

David Choffnes, John Lange, Sam Rossoff, and Aleksandar Kuzmanovic

Abstract

Network measurements reveal that today's most popular web browsers open parallel TCP connections and use them to actively transfer data from web servers. This technique, based on the assumption that ``more is better,'' is motivated by the need to improve user-perceived performance. While parallel connections indeed improve performance for lossless networks and when web pages are very large in size, we show, by means of analytical modeling, simulation, and testbed experiments, that no such improvements exist in scenarios that more closely characterize today's Internet. Moreover, we demonstrate that in addition to placing more stress on web servers, the parallel TCP approach can degrade the web-response times to a level that is up to an order of magnitude below the level achievable by a single connection. We analyze the roots of this phenomenon and find that the benefits of accelerated parallel download of small objects are largely overshadowed by the initial connection setup time, which dominates the entire transfer. We quantify the user-perceived latency and show that it dramatically increases with the level of parallelism and congestion in the network.

Keywords: Web; parallel connections; TCP; HTTP; performance

1 Introduction

Despite the fact that several recently-proposed TCP versions demonstrated a clear ability to successfully utilize the available bandwidth on an end-to-end path (*e.g.*, (1; 2)), applications using *parallel TCP* connections are thriving in the Internet. For example, the vast majority of peer-to-peer users employ the parallel File Transfer Protocol (FTP) (3; 4), initially developed to improve efficiency of large data transfers in grid-computing environments. While such applications technically do not violate fairness in the network (because all flows applied in parallel download are TCP-friendly), this selfish behavior is unfair to those who use a single TCP connection to transfer data.

A similar trend is observed in the Hypertext Transfer Protocol (HTTP). Contrary to parallel FTP, where both endpoints must collaborate in partitioning and transferring a file over parallel TCP connections, HTTP does not require such cooperation from web servers. This is because pages on a web server are typically fragmented; *i.e.*, they consist of a number of small objects that can be fetched by separate connections.

Historically, the problem of page fragmentation caused significant changes in the HTTP specification. In its first incarnation, HTTP/1.0 (5), the client and server open and close a new TCP connection for each object, which causes significant degradation in client-perceived response times (6). To address this problem, HTTP/1.1 (7) enables a *persistent* connection with *pipelining*: the TCP connection remains open (persistent) between consecutive operations, and allows multiple HTTP requests to be sent without waiting for a response (pipelining). It has been demonstrated that such an approach significantly improves the client-perceived response times (8; 9; 10).

Despite the fact that HTTP/1.1 specification *recommends* opening up to two persistent TCP connections,[1] today's most popular web browsers employ a larger number of *parallel persistent* TCP connections in an attempt to improve performance when downloading pages contain multiple objects. Indeed, measurements from (11) reveal that both `Internet Explorer` and `Netscape` open up to seven parallel persistent TCP connections to a web server, and use them concurrently to transfer data. While opening multiple connections to different servers makes perfect sense in presence of distributed web content (*e.g.*, Akamai (12)), *evidence show that browsers in addition open parallel connections to each such server* (11).

This design choice is motivated by a need to improve the client-perceived response times. For example, the use of parallel connections could result in a larger aggregated window size. Also, in case of packet drops, the parallel transfer could experience a less aggressive backoff, provided that only a single TCP flow from the aggregate will halve its window size. Similar to the parallel FTP case, this approach aims to optimize web browser performance without concern for how this can affect others, and without worrying about the additional stress placed on the servers.

In this paper, we show that parallel TCP connections largely lack the capacity to improve the client-perceived response times in scenarios with realistic packet loss

[1] RFC 2616: "A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy." Thus, opening more than 2 connections is not strictly forbidden.

ratios and web page sizes. Moreover, we demonstrate that the parallel-connections approach can degrade client performance to a level that is significantly *below* that of a single persistent TCP connection. The key reason for this effect is the gap between the cost of setting up the parallel connection and the benefit of accelerated parallel download. While it may appear obvious that establishing a larger number of TCP connections incurs a larger overhead, the key observations is that the cost of establishing parallel connections grows considerably with the number of connections.

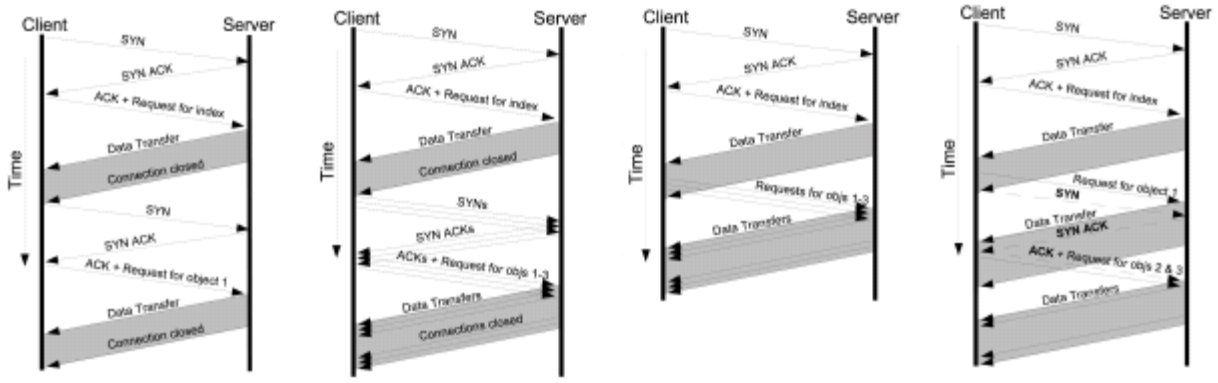
Consider a single TCP connection first. If TCP control packets (SYN or SYN ACK) are lost during the connection setup, the endpoints must wait during extremely long initial timeouts before reattempting to establish a connection. In this case, the client-perceived response time is dominated by this initial setup time because the actual data transfer time is much shorter than the connection establishment time.

When parallel TCP connections are used, the expected connection establishment latency increases with the number of connections. The larger the number of parallel connections, the larger the probability that TCP control packets will be lost, and the larger the expected connection-establishment latency. At the same time, a larger number of established TCP connections necessarily reduces the data-transfer latency. However, when the objects are small in size, the decrease in the data-transfer latency (relative to the single-connection case) is much smaller than the increase in the connection-establishment latency. Thus, overall parallel-connection performance worsens compared to a single connection.

We motivate this problem by developing an analytical model, generalizing (13; 14) to predict the client-perceived latency as a function of the number of parallel connections and the packet loss ratio. We develop a model that accounts for connection establishment latency with parallel connections and the time spent in TCP's data transfer phase. We use this model to predict the total time to download fixed-size pages with various numbers of parallel connections and packet loss ratios.

Next, we perform a set of large-scale simulations. In our first set of experiments, we validate the model by showing that, on average, a single connection outperforms parallel connections for relatively small file sizes, even at low levels of packet loss. As packet loss increases, the penalty for parallel connections is up to an order of magnitude. We then apply a workload that reflects a real-world scenario to determine the effect of parallel connections on a typical web browsing experience. We show that for this realistic scenario, a single connection offers the best average performance and that parallel connections extremely rarely improve response times.

Finally, we perform a set of testbed experiments to verify our findings in a real system. The results of these experiments further validate our model and help to prove that parallel persistent connections should be used with caution in today's web browsers. At the end, we discuss several related issues: the impact of distributed



(a) HTTP/1.0, no parallel connections

(b) HTTP/1.0 with parallel connections

(c) HTTP/1.1 with pipelining

(d) HTTP/1.1 with pipelining and parallel connections

Fig. 1. Diagram depicting the dynamics of TCP connections when using nonpersistent, persistent, parallel and pipelined connections.

web content, the server-side effects, the opportunistic use of parallel TCP connections, and the importance of the whole page performance for web applications.

2 Background

In this section, we provide an overview of the HTTP specifications. We first briefly explain the components of TCP that are most critical to HTTP performance. Then, we focus on differences in the ways that HTTP variants establish and use TCP connections. In the following discussion, we consider a scenario in which a client (web browser) establishes connections to a server (web server).

To initiate a TCP connection, the client sends a SYN packet to the server, indicating that it wants to establish a TCP connection. When the server receives the packet and accepts the connection, it replies with a corresponding SYN ACK packet. Finally, the client acknowledges the SYN ACK with an ACK packet to fully establish the connection. This is called the *three-way handshake*. After a connection has been fully established, TCP uses timers and ACKs to determine whether packets for the connection have been lost, and retransmits those packets that have not been explicitly acknowledged as received.

Figure 1 depicts specific ways in which HTTP variants establish and use TCP connections. Despite their differences, all versions use the same technique to initially request and download the index page of a given web page. An index page contains references to the objects embedded in the corresponding web page. As indicated in the figure, the client piggybacks the HTTP request to the ACK packet of the initial

three-way TCP handshake. Upon receiving the HTTP request, the server replies with the index page. Depending on the HTTP version, the objects referenced in the index page are fetched from the server in the following ways.

The HTTP/1.0 specification uses nonpersistent TCP connections (5). As indicated in Figure 1(a), this means that a new TCP connection must be established for each object retrieved from the server. Thus, the HTTP/1.0 client closes the initial TCP connection upon the receipt of the index page; then, it opens a new TCP connection for the first embedded object, retrieves the object, and closes the TCP connection. The client repeats this procedure for each of the objects from the page in a serial fashion, as indicated in Figure 1(a). Clearly, when a web page contains multiple embedded objects, performance in terms of aggregate latency (*i.e.*, the time from the first request for a page until the time when the last object is downloaded), significantly degrades due to the overhead suffered during multiple connection establishments. Also, because most web objects are small, TCP rarely leaves the slow start phase, further limiting performance.

To improve HTTP/1.0 performance, many HTTP/1.0-enabled web browsers establish multiple parallel connections to retrieve objects. Indeed, once the client receives the index page that references all embedded objects, nothing stops the client from establishing multiple TCP connections and retrieving objects in parallel from the server. The parallel TCP approach was never explicitly addressed in the HTTP/1.0 specification; instead, it was assumed to be part of a web browser's optimizations. As shown in Figure 1(b), opening parallel connections allows the client to incur multiple connection-establishment overheads "at the same time," reducing aggregate latency.

Another approach to solving the limitations of the initial HTTP/1.0 specification is the use of persistent TCP connections as proposed in HTTP/1.1 (7). As depicted in Figure 1(c), the HTTP/1.1 client need establish only one connection with a server to retrieve an entire web page and its embedded objects. In addition, HTTP/1.1 also enables *pipelining*, where a client uses a persistent TCP connection to request multiple objects in succession. Each object is requested without waiting for the previous request to complete, which enables clients to eliminate one RTT latency per downloaded object.

Finally, while HTTP/1.1 specification "softly" recommends opening up to two persistent connections,[2] this does not prevent browsers from establishing multiple parallel persistent connections (11). In addition, when pipelining is enabled — the case explored in this paper — these become parallel persistent pipelined connections.[3] This scenario is shown in Figure 1(d). In this example, the client establishes a connection to the server to retrieve the web page, then requests embedded objects

[2] In the RFC terminology, SHOULD correspond to a "soft," and MUST to a "hard" recommendation.

[3] Not all browsers support pipelining by default. We discuss this issue in Section 7.

using the existing connection and a new one that it opens in parallel.

Initially, this implementation appears to retain the improvements established in the HTTP/1.0 case. In addition, it appears that it outperforms the single persistent HTTP/1.1 case (Figure 1(c)), provided that the use of parallel connections results in a larger aggregated window size, thereby improving user-perceived performance. However, the parallel connections approach requires both the client and the server to send a larger number of TCP control (*e.g.*, SYN and SYN ACK) packets, which increases the probability that at least one such packet will be lost in the network. A loss of a single TCP control packet invokes initially a 3-sec long timeout, which doubles for each successive loss. Thus, when the web objects are small in size, the loss of TCP control packets can dramatically increase the connection-establishment latency; this can negate the potential benefits of accelerated data transfer that occurs due to larger number of connections. We treat these problems in depth in the following sections.

3 Modeling

In this section, we develop an analytic model to evaluate the effects of parallel connections on user-perceived web-response times. Our analysis builds upon the work of Cardwell *et al.* (13), which calculates the expected latency for a single TCP connection over a lossy link. Our key contribution here is the generalization of this model for the parallel connections scenario.

We follow the established terminology and assumptions from (13; 14). In addition, we assume that the packet loss ratio in the direction from the server to the client is dominant. This is justified by the fact that the congestion is much more likely to arise in the server-to-client direction due to a larger amount of traffic being downloaded by clients. Also, we assume that the index page size is small enough to fit inside a single TCP packet. This can slightly underestimate the user-perceived latency, as we explain in detail below. Both of the assumptions considerably simplify our analysis; later in the paper we evaluate the model and show that these assumptions do not significantly impact the model's accuracy.

We consider the total time to download a web page as the key performance metric (15). Although web clients often attempt to render web pages even before the entire index page has been downloaded, this traditionally yields poor results for media-rich pages. For example, if the web page includes images essential to the user interface (*e.g.*, images for input buttons) or dynamic content such as Flash applications, Javascript code or Java applets, the user-perceived latency must encompass the time until these objects have been fully downloaded. Finally, we assume that the browser is not using any "opportunistic" schemes, i.e., sending HTTP request for the same object over multiple connections, or opportunistically discarding TCP connections

that experience congestion problems. To the best of our knowledge, web browsers do not apply such techniques. We revisit this issue in Section 7.

3.1 Connection Establishment

Figure 1(d) shows that the client and the server perform a three-way handshake to establish the initial TCP connection. At each stage of this process, if either party does not receive the ACK that it is expecting within some retransmission timeout (RTO) seconds, it retransmits its SYN and doubles the RTO value. According to RFC 2988 (16), the RTO is initially set to three seconds. [4] Because we assume that packet losses happen in the server-to-client direction, we model the losses of SYN ACK and other packets generated by the server.

Denote the packet loss ratio in the server-to-client direction by p. Let $P_h(i)$ be the probability of a three-way handshake episode consisting of exactly i failures transmitting SYN ACK packets, followed by one successful SYN ACK packet. Then

$$P_h(i) = p^i(1-p). \tag{1}$$

Denote by $L_{est}(i)$ the latency experienced by the client to receive a SYN ACK packet. This latency can be expressed as

$$L_{est}(i) = RTT + \left(2^i - 1\right) RTO. \tag{2}$$

Indeed, if the initial SYN ACK packet successfully reaches the client, the latency equals RTT. If it gets dropped in the network, the latency becomes $RTT + RTO$, etc. Thus, the expected latency to receive a SYN ACK packet, $E[L_{est}]$, becomes

$$E[L_{est}] = \sum_{i=0}^{\infty} P_h(i) L_{est}(i). \tag{3}$$

Next, the client sends a request for the index page to the server, which replies with a single packet containing the index page. This packet experiences the same congestion as the SYN ACK packet. Moreover, according to RFC 2988, the initial RTO is again set to 3 sec for the first data packet. Thus, the *expected value* of the latency experienced by the client to receive the index page packet after it has received the SYN ACK packets equals the above expected latency for the client to initially receive a SYN ACK packet. Hence, the expected value of aggregate

[4] Our measurements confirm that FreeBSD and Linux boxes do follow this recommendation.

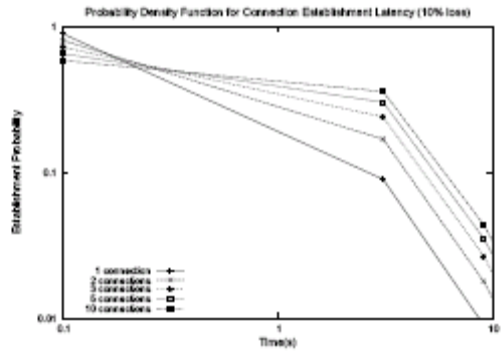


Fig. 2. Modeling: $P(i, n)$ as a function of $L_{est}(i)$.

latency for the index page, $E[L_{ind}]$, from sending the SYN packet to downloading the index page, becomes $2E[L_{est}]$.

Once the index page is downloaded, the client initiates a number of parallel connections. We assume that the connection establishment phase finishes when *all* TCP connections are established.[5] Thus, the entire process is dominated by the slowest connection. Denote by n the number of parallel connections. Next, denote by k, $k \leq n$, the number of "slowest" connections; the connections from this group experience exactly i failures transmitting SYN ACK packets, while the rest of the $n-k$ connections experience a smaller number of failures. Denote by $P(i, n)$ the corresponding probability. Then

$$P(i,n) = \sum_{k=1}^{n} \frac{n!}{k!(n-k)!} \left[P_h(i)\right]^k \left[\sum_{r=0}^{i-1} P_h(r)\right]^{n-k}, \tag{4}$$

for $i \geq 1$, while $P(0, n) = (1-p)^n$. In other words, the probability that all connections are successfully established without a failure transmitting SYN ACK packets is $(1-p)^n$. On the other hand, the probability that k connections experience i failures equals $[P_h(i)]^k$; the probability that $n-k$ connections experience less than i failures is $[\sum_{r=0}^{i-1} P_h(r)]^{n-k}$. Finally, the term $\frac{n!}{k!(n-k)!}$ counts all possible combinations of the above events.

Figure 2 plots the connection-establishment probability for n parallel connections, $P(i, n)$, as a function of latency $L_{est}(i)$, for i =0, 1, and 2. Thus, the relevant points on the x-axis are RTT (0.1 sec), $RTT+RTO$ (3.1 sec) and $RTT+3RTO$ (9.1 sec). The key point from the figure is that the probability of experiencing a larger number of retransmissions increases with the number of parallel connections. For example, the probability that ten connections are established within RTT sec is lowest relative to other scenarios in which a smaller number of connections is used. This is

[5] A TCP connection can be used to fetch web objects as soon as it is established, without waiting for other connections to finish the setup process. We explain this in more detail below.

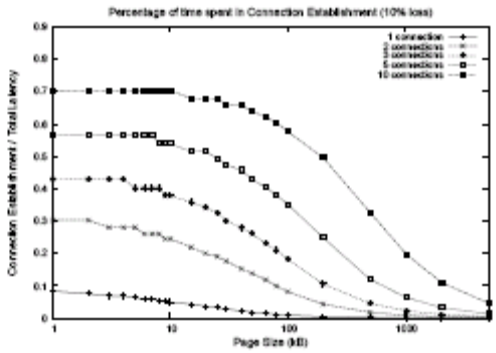


Fig. 3. Modeling: The ratio of connection-establishment and total latencies.

shown in Figure 2 for the 0.1 sec latency. At the same time, the probability that at least one of the connections experiences exactly one retransmission is the highest in the case of ten connections, and the smallest in the case of a single connection, as indicated in the figure for the 3.1 sec latency. This affects the expected parallel connection establishment latency, as we explain below.

Finally, denote by L^n_{est} the connection-establishment latency for n parallel connections, from sending the SYN packet for the index page to receiving the SYN ACK packets for all n connections. Then, the expected value of L^n_{est}, $E[L^n_{est}]$, can be expressed as

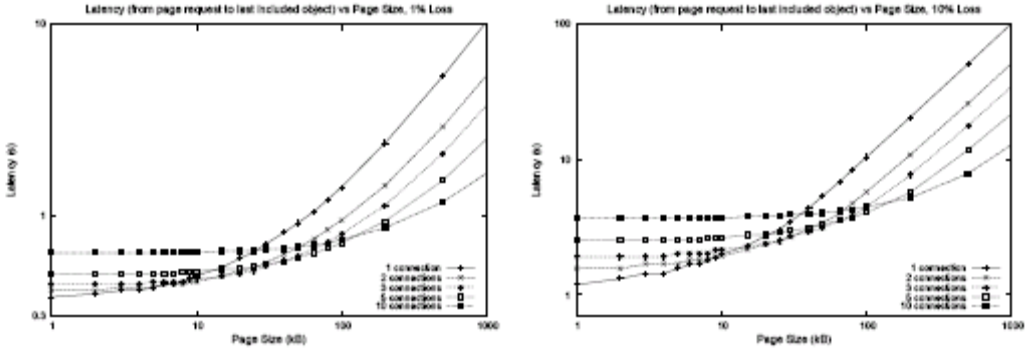
$$E[L^n_{est}] = E[L_{ind}] + \sum_{i=0}^{\infty} P(i, n-1) L_{est}(i). \tag{5}$$

Equation (5) indicates that the expected latency to establish n parallel connections equals the sum of the expected latency to establish the initial TCP connection and download the index page, $E[L_{ind}]$, and the expected latency to establish the remaining $n-1$ connections.

3.2 Total Time to Download

To fully understand the impact of establishment latency on the web transmission transmission as a whole, we need a method of modeling the expected latency of the actual data transfer. Here, we make no attempts to develop such a model; instead, we apply the well-established results from (13). In (13), the authors model the TCP slow start and congestion avoidance phases to compute the expected value of the time spent in data transfer, $E[L_{dt}(D)]$ (Equation (25) from (13)), as a function of the file size D. Below, we use this result to compute the expected latency when parallel connections are used.

Denote by $L^n_{tot}(D)$ the total time needed to download a web page of size D with n



(a) 1% packet loss (b) 10% packet loss

Fig. 4. Modeling: The page response time versus the page size for different levels of packet loss.

parallel connections. Then, the expected value of $L^n_{tot}(D)$ becomes

$$E[L^n_{tot}(D)] = E[L^n_{est}] + E[L_{dt}(D/n)]. \tag{6}$$

Equation (6) essentially breaks the connection lifetime into two separate stages: connection establishment and data transfer. This equation appears to indicate that we are forcing all connections to establish before transferring any data, which is *not* the case. Equation (6) indicates that the last connection to establish dominates the whole page latency. This is because each TCP connection is responsible, ***on average***, for transferring the same amount of data, D/n. This is a valid assumption because there is no evidence that web browsers use intelligent connection assignment such that the first connection to establish is given the largest object to download. Moreover, this is currently impossible because the HTTP index pages do not provide information about object sizes. In addition, given that all connections share the same network path, we assume that each connection experiences the same rate of packet loss and the same connection bandwidth. Therefore, we contend that the expected value of the total time to download a web page is governed by the last connection to successfully establish. Both our simulation and testbed experiments prove this assumption to be valid.

Figure 3 depicts the ratio of the expected values of the connection-establishment and the total latencies ($E[L^n_{est}]/E[L^n_{tot}(D)]$), as a function of the page size D, for different number of parallel connections n. The key observation is that for small to medium page sizes, the connection establishment latency represents a large portion of the total latency when parallel connections are used. For example, for the case with a page size of 100 kB and one connection, the establishment latency represents only a small percent of the total latency. On the contrary, it represents as much as approximately 60 % of the total latency when ten connections are used in parallel.

Finally, Figure 4 plots the expected latency, $E[L^n_{tot}(D)]$, as a function of the page

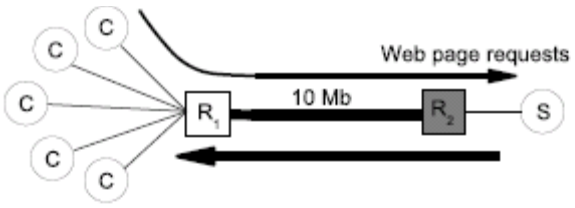
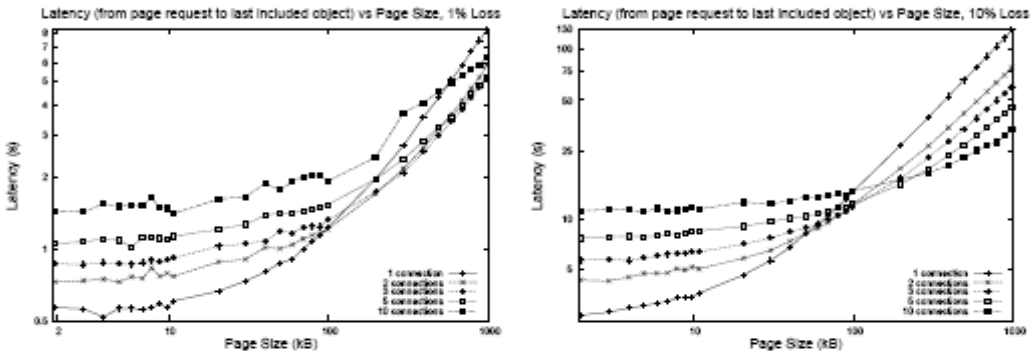


Fig. 5. Simulation scenario



(a) 1% packet loss (b) 10% packet loss

Fig. 6. Simulation: The page response time versus the page size for different levels of packet loss.

size D, and for different number of parallel connections n; the packet loss ratios are 1% and 10%. The figures indicate that for small to medium page sizes, the use of parallel connections can degrade the total latency to a level that is *below* the level achievable by a single connection. This is because the connection-establishment latency increases with the number of parallel connections used. Since this latency dominates the total latency for small to medium page sizes, the single connection achieves the best performance. Figures 4(a) and 4(b) further show that this effect is more pronounced for larger packet loss ratios. This is due to fact that the connection establishment latency increases with the packet loss ratio. Finally, the figure indicates that as the page size increases, the benefits of parallel download start to dominate the total latency. In the next section, we explore these effects via simulation.

4 Simulation Experiments

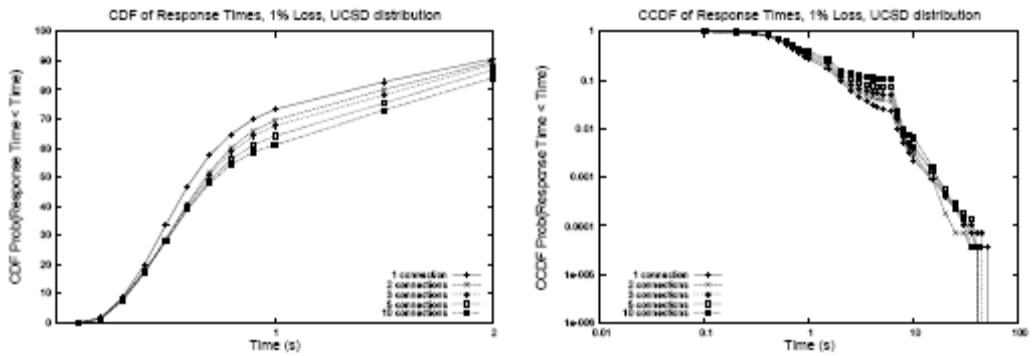
4.1 Simulator Setup

Figure 5 depicts our simulation scenario. It consists of a web-client pool and a web server, which are interconnected by a pair of routers and a bottleneck link. Each node from the client pool connects to router R_1 via a 1 Gbps link; likewise, the web server and router R_2 are connected by a 1 Gbps link, while we set the capacity of the link between routers R_1 and R_2 to 10 Mbps. We uniformly distribute the flow round-trip times in the range from 10 ms to 100 ms. In order to accurately control the congestion level at R_2, we randomly drop p percent of packets that pass through the router from the server to the clients. This is accomplished using the *ns-2* extensions from (17). We set the congestion in the server-to-client direction simply because this is more likely to happen. Fundamentally, there should be no difference if the congestion is more dominant in the reverse direction.

In the experiments, clients send HTTP requests and retrieve web pages from the server. We set the index page size such that it can fit in one packet and thus minimally affect the aggregate latency. The object sizes, as well as the number of embedded objects per page vary in different experiments; we provide the exact values below. We use a TCP window size of 16 kBytes and a segment size of 1460 Bytes. [6] In order to fully explore the design space, we change the maximum number of parallel connections among 1, 2, 3, 5 and 10. Also, we vary the packet loss ratio between 0 and 10%. Our goal is to emulate both moderate congestion scenarios and heavier ones that may arise due to flash-crowd events (19). Network measurements reveal that the median per-connection packet loss ratio in the Internet is 1% (20). However, a non-negligible percent of clients may experience high packet loss ratios, *e.g.*, above 10% (21).

We upgraded the extension from (22) to enable the support of parallel persistent TCP connections as follows. When a client requests a page, it establishes a persistent TCP connection to the server and sends the HTTP request, which is encapsulated in a single fixed-size TCP packet. After receiving the index page, the client establishes the maximum allowed parallel connections for the simulation run, then retrieves the embedded objects in a pipelined manner. If the number of files embedded in a page is greater than the maximum number of parallel connections, file requests are pipelined round robin into the available connections. If the number of embedded objects is smaller than the maximum number of parallel connections for the run, then the client opens enough connections such that there is one connection per object. Indeed, reference (23) indicates that this is the policy currently implemented by web browsers.

[6] The measurement study from (18) reveals that the median advertised window parameter in the Internet is 16 kBytes.



(a) CDF (b) CCDF

Fig. 7. Simulation: CDF and CCDF response-time profiles; packet loss rate is 1%.

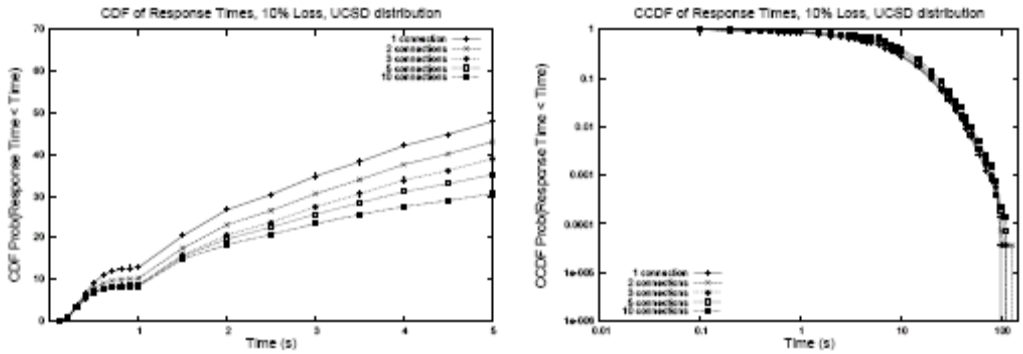
4.2 Evaluation

4.2.1 The User-Perceived Latency as a Function of the Page Size

We begin the simulation evaluation by examining the effects of the page size and packet loss on web browser performance. Here, the key performance measure is the end-to-end response time for each *page* request/response pair. For a given page, we compute its response time from the moment when the first request for the index page is sent to the server, until the whole page is successfully downloaded by the client.

Figure 6 plots the average response time (the y-axis in the figure) as a function of the page size (the x-axis in the figure) for different maximum numbers of parallel connections allowed. For each page size value, the web pages are made to contain a number of objects between one and the maximum number of parallel connections; we collect 1000 latency values for each such run. Each data point in Figure 6 represents the average over all of these runs, and the standard deviation (not shown) tends to be quite large for smaller aggregate data sizes. This occurs due to the large timeout interval during connection-establishment. Nevertheless, the averages quite successfully capture the differences induced by the varying number of parallel connections.

Figure 6(a) demonstrates the effect of varying the number of parallel TCP connections with 1% packet loss, representing a scenario with a moderate packet loss ratio. We use a log-log scale to emphasize the change in latency over small page sizes. Figure 6(a) shows that latency is better with one connection for small-to-medium page sizes, with multiple parallel connections becoming effective only when the page size is greater than 100-500 kB. For example, two parallel connections outperform the single connection approach for page sizes larger than 100 kB, while 10 connections achieve the same for page sizes larger than 500 kB. This is because



(a) CDF (b) CCDF

Fig. 8. Simulation: CDF and CCDF response-time profiles; packet loss rate is 10%.

the connection setup latency increases with the number of parallel connections, as established in the previous section.

Figure 6(b) repeats the previous experiment, but using 10% packet loss, representing a scenario with a highly congested server. In this case, the difference between a single connection and multiple connections is more pronounced for short pages. For example, there is an order of magnitude improvement in latency for a single connection compared to 10 parallel connections. This is because the connection setup latency increases with the packet loss ratio and the number of parallel connections. On the other hand, the parallel connections approach start to "pay off" sooner, for shorter page sizes. For example, 10 connections outperform a single connection for page sizes larger than 140 kB; this is smaller than 500 kB required in the 1% loss scenario. Thus, despite the fact that parallel connections initially lag more dramatically behind a single connection, they manage to improve the overall performance and outperform the single connection sooner; this is due to extremely poor performance of a single TCP connection in highly congested environments (1; 2).

4.2.2 Response-Time Distribution Profiles

While quite insightful, the above experiment does not use a realistic object- or page-size distribution; instead, it uniformly distributes a given page size on a number of objects. In the next set of experiments, we apply a workload that reflects a real-world scenario to determine the effect of parallel connections on a typical web browsing experience. In particular, the object sizes are set according to the distribution reported in (17), which is obtained from representative network measurements. Similarly, the number of embedded objects per page is set according to a distribution reported in (11), which is derived based on a representative sample consisting of over 22,000 web objects.

In order to express the above effects in a uniform manner, we proceed as follows. We report the cumulative distribution function (CDF), $F(x) = Pr[X \leq x]$, of response times up to an upper bound (*e.g.*, 2 or 5 seconds), as originally proposed in (24). CDF plots focus on the short-latency region where the vast majority of pages are downloaded, and they accurately reflect differences that arise due to parallel connections. In addition, we report the complementary cumulative distribution function (CCDF), $1 - Pr[X \leq x]$, of response times; the CCDF profiles capture the performance of large pages and best reveal the tails of response-time distributions. In these graphs, each data point represents the average of over 12,000 web page downloads.

Figure 7(a) plots the CDF curves for average latency with 1, 2, 3, 5 and 10 parallel connections and 1% packet loss. The figure clearly shows that, on average, using a single persistent connection results in lower latencies than using multiple connections. For example, approximately 75% of the pages are downloaded in less than a second when a single connection is used, while this percentage drops to 60% when 10 parallel connections are used. As latency increases, the separation between the curves decreases. This happens because, as object sizes increase, parallel download becomes more effective. Overall, however, parallel downloading never manages to outperform the single connection approach.

The CCDF plot in Figure 7(b) further confirms this result. The smaller the tail of the distribution is, the smaller the mean response time, and the better the performance of a particular scheme. Although the curves are very similar, the single-connection approach still retains the smallest tail. This is because a larger number of large objects in the same page (a scenario in which parallel connections outperform the single connection case) rarely occurs.

Finally, Figure 8(a) repeats the previous experiment, but changes the packet loss to 10%. Still, the single connection has the best performance. Figure 8(b) further shows that the CCDF profiles for different connections are almost identical. Thus, despite the potential for parallel connections to improve the response times of large pages, such events are very rare.

5 Testbed Experiments

Here, we perform a set of testbed experiments with the goal of verifying the above findings in a real system. We replicate a topology similar to the one depicted in Figure 5. Our testbed consists of a server and a client machine. The server machine runs Linux 2.6 and the Apache 2.0 web server (25); the index page is implemented as a dynamic web page that generates objects of specific sizes. The client machine runs FreeBSD and a simple web client program that we explain in more detail below. Both machines are Dell Dimension Desktops with Pentium IV 3.0 GHz processors,

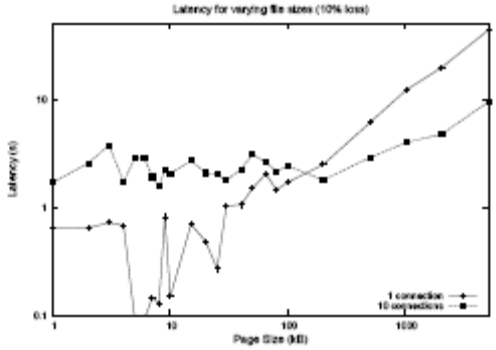


Fig. 9. Testbed experiments: The page response time vs. the page size

and 100 Mbps network interface cards. Provided that the congestion is typically more likely to happen in the direction from the server to the clients, we use the Traffic Control (tc) utility to probabilistically drop packets from the sending queue on the server.

Our web client implementation consists of a program that initially creates a single connection to the server and then downloads a small index page. [7] The index page size is small enough to fit into a single packet. Following the index page download, the client program opens the maximum allowed parallel connections for the experimental run; then, it retrieves the embedded objects in a pipelined manner as explained in the previous section. The initial connection used to download the index page is reused as one of the parallel connections downloading the referenced objects. The client program spawns multiple threads to handle the parallel connections. Our measurements show that processing time has little to no effect on the behavior of the network connections, so we do not include it in our analysis.

Figure 9 shows the average page response time as a function of the page size, both for a single pipelined connection and for 10 parallel connections. The packet loss rate is set to 10%. Due to the smaller number of latency values compared with the simulation experiments, the curves in Figure 9 are not as smooth for smaller page sizes. As explained, this large deviation occurs due to large timeout interval during connection-establishment. Indeed, while Figure 9 may leave the reader with the impression that the difference of several hundreds of *ms* between the two curves may not be relevant for clients, ***this is not the case.*** Figure 9 depicts the *average* latency; *it reveals that the number of long (and annoying) TCP timeouts is significantly larger when parallel connections are used.*

The curve for 1% loss ratio, while showing the same general trend, exhibits even larger variations, so we refrain from showing it. Nevertheless, our results line up reasonably with our modeling and simulation results.

[7] The source code is available at `http://www.cs.northwestern.edu/~drc915/webBrowsPerf/`.

6 Related Work

Web performance has been thoroughly evaluated over the last decade. This section presents a detailed review of the relevant work concerning HTTP performance.

Since persistent HTTP (P-HTTP) was first proposed by Mogul (6; 26), and since its subsequent implementation in HTTP/1.1 (7), the evaluation of this technique has been a popular research topic. Nielsen *et al.* (8) showed that a pipelined HTTP/1.1 implementation outperforms HTTP/1.0, even when the HTTP/1.0 implementation uses multiple connections in parallel (Figure 1(b)). In this paper, we showed that a HTTP/1.1 implementation that uses a *single* pipelined TCP connection significantly outperforms the parallel pipelined HTTP/1.1 approach.

Next, Krishnamurty and Wills (9) demonstrate that pipelining and persistence improves relative performance of pages with more objects. They explore single persistent HTTP/1.1 connections with pipelining versus serial HTTP/1.0 connections and show that HTTP/1.1 offers the best performance. Farkas *et al.* reach the same conclusion in (10).

On the contrary, the measurements by Barford and Crovella (27) indicate that pipelining does not significantly reduce file-transfer latency. They conclude that this difference is due to the shorter RTT times in their LAN environment. Similarly, Kruse *et al.* (28) point out the problem of ordering, retrieval, and display of page elements in HTTP/1.1 that can cause the user's perceived performance to degrade. Finally, Heidemann (29) finds and solves several performance problems caused by interactions between P-HTTP and TCP.

To the best of our knowledge, Bent and Voelker (15) are the first to evaluate the impact of *parallel persistent* connections on client-perceived performance. Contrary to our findings, they conclude that enabling parallelism improves the user-perceived performance. The inconsistency between the two results is due to different evaluation environments. Bent and Voelker generated their measurements by replaying browsing traces; but, as pointed out in (15), these measurements were gathered *overnight*, which means in a highly uncontested environment. Indeed, if there is no congestion in the network, the price of establishing multiple connections is the same as the price of establishing a single connection but the increased number of connections accelerates the data transfer. Thus, enabling parallelism improves the user-perceived performance in such cases, as our model predicts. However, we showed that the results change dramatically when the level of parallelism and the packet loss ratio increase.

Balakrishnan *et al.* (30) study the effect of parallel connections on end-to-end performance and the network. They conclude that existing loss recovery techniques are not effective for avoiding timeouts and that a client using a collection of parallel connections is more aggressive. Hence, they propose integrated congestion

control/loss recovery scheme. Despite the fact that they motivate their work by web server behavior, they apply and evaluate their solution in scenarios with *long-lived* TCP flows. In such cases, the connection establishment problems are less pronounced.

Others have also treated the problems related to the performance of parallel TCP connections in a wide-area network environment (3; 31; 32). Particularly interesting is the work of Lu *et al.* (31), where the authors develop a model to allow applications using parallel TCP connections to increase their aggregate throughput without placing undo stress on the network. Our work derives a model for small parallel TCP flows, in which the setup cost is the dominant factor of performance.

Finally, the sensitivity of end-to-end performance to loss of TCP *control* packets has been explored previously. In (33), Hall *et al.* measure data loss to a news server. The results clearly show that the largest performance degradations are due to loss of SYN packets.

7 Discussion

Distributed web content. In today's Internet, many web objects are fetched from multiple servers. This is because many popular web sites are hosted by content distribution network (CDN) providers (*e.g.*, Akamai (12)). In such scenarios, the browser opens separate TCP connections to each of the origin servers and fetches data concurrently. While parallel connections make perfect sense in such cases, in this paper, we focused on a different problem — the one in which a web browser opens parallel TCP connections to the *same web server*. Measurements reveal that in addition to opening parallel connections to different origin servers, browsers open multiple connections to each such server (11).

Server-side effects. This paper focuses on network-side effects of the examined problem. While server-side effects are certainly relevant, they are beyond the scope of our work here. Nevertheless, the use of parallel connections increases the number of TCP SYN packets send to the server, which necessarily magnifies the burden placed on it. While modern servers are becoming increasingly resilient to such stresses (34), reducing the number of TCP SYN packets could only improve a server's performance.

Opportunistic use of TCP connections. To the best of our knowledge, modern web browsers are not opportunistically using TCP connections. For example, a browser may send a number of TCP SYN packets and then simply use the one whose SYN ACK packets arrives the first. Anderson *et al.* (35) demonstrated (in a somewhat different context) that such an approach can improve availability of web services. While the same approach would be capable of improving clients' perfor-

mance in the classical web client/server scenario, adoption of such a technique by popular web browsers would place an enormous additional load on servers, *e.g.*, incomparably larger than what a small-scale overlay testbed from (35) is capable of. Likewise, while it may appear attractive to remove the RTO mechanism all together, pursuing this avenue is dangerous, as timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion (36).

Pipelining. Like almost any innovation in the Internet, the deployment of pipelining is experiencing difficulties. This is mostly due to "broken" load-balancing proxies that do not handle pipelining properly. As a result, web browsers either disable pipelining by default and leave the final decision to clients (*e.g.*, `Netscape` and `Firefox`) or implement heuristics to enable/disable it (*e.g.*, `Opera`). While it is clear that pipelining is essential for achieving high page-load performance, our key contribution lies in demonstrating that it is *not* sufficient: in addition to pipelining, browsers must refrain from opening multiple connections.

The whole-page metric. We use the *average* time to download the *whole web page* as the key performance metric to demonstrate the advantages of a single-connection approach. One criticism of such a metric is that it doesn't consider the so-called head-of-line blocking problem in the single-connection scenario; if the TCP connection is broken — all objects are stalled. However, this does not change our result which states that the single connection approach *on average* outperforms parallel connections. A good analogy can be made with air transportation; when an airplane crashes, chances are very slim that any of the passengers will survive the crash. However, on average, airplanes are by far the safest means of transportation.

Even though users are ultimately most interested in whole-page performance (15), another criticism is that this metric is not relevant because clients may parse partially loaded pages. Although this technique enjoys limited success for reducing user-perceived latency, there are many scenarios in which the whole-page metric is essential. For example, a web page that relies on a Flash application or Java applet for its core functionality cannot be considered fully loaded until the requisite embedded objects have been fully loaded.

8 Conclusions

This paper examined the effects of the use of parallel connections in web browsers on client performance and discussed the effects that this technique has on server performance. We showed that for small to medium page sizes, the parallel connections can degrade the user-perceived latency by up to an order of magnitude relative to the single connection case. The key reason for this effect is the gap between the cost of setting up the parallel connections and the benefit of accelerated parallel

download. The high establishment cost is induced by TCP control packet losses that cause long retransmission timeouts; the benefit of parallel download is small when the web objects are small in size.

We developed an analytic model that successfully captures several key system dependencies. In particular, the model quantifies the way in which the connection-establishment latency increases with the number of connections and the level of congestion. We bolstered our findings by simulation and testbed experiments and showed that, depending on the congestion level and the number of parallel connections used, parallel download starts to outperform the single connection approach when the page size is beyond 100 - 500 kB. Finally, we evaluated realistic web workloads to determine the impact of parallel connections on typical web browsing scenarios. While it is clear that opening multiple connections certainly places additional stress on web servers, our results show that this technique *on average* does more harm than good to clients, independently of the server load. Thus, *all* clients have incentive to abandon this well-established practice; not only for the benefit of servers, but for their own sake.

References

[1] S. Floyd, J. Madhavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," July 2000, Internet RFC 2883.

[2] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of ACM MOBICOM '01*, Rome, Italy, July 2001.

[3] T. Hacker, B. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *Proceedings of IPDPS '02*, Fort Lauderdale, FL, Apr. 2002.

[4] J. King, "Parallel FTP performance in a high-bandwidth, high-latency WAN," Feb. 2001. [Online]. Available: `http://www.llnl.gov/asci/discom/sc2000_king.html`

[5] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol - HTTP/1.0," May 1996, Internet RFC 1945.

[6] J. Mogul, "The case for persistent-connection HTTP," in *Proceedings of ACM SIGCOMM '95*, Boston, MA, Aug. 1995.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - HTTP/1.1," June 1999, Internet RFC 2616.

[8] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, "Network performance effects on HTTP/1.1, CSS1, and PNG," in *Proceedings of ACM SIGCOMM '97*, Cannes, France, Sept. 1997.

[9] B. Krishnamurthy and C. Wills, "Analyzing factors that influence end-to-end

web performance," in *Proceedings of WWW '00*, Amsterdam, Netherlands, May 2000.

[10] K. Farkas, P. Huang, B. Krishnamurthy, Y. Zhang, and J. Padhye, "Impact of TCP variants on HTTP performance," in *High Speed Networking Conference*, Budapest, Hungary, May 2002.

[11] L. Bent, M. Koletsou, and G. Voelker, "The Medusa proxy: Parallel connections under two browsers," Aug. 2002. [Online]. Available: `http://ramp.ucsd.edu/medusa/parallel.html`

[12] "Akamai," [Online]. Available: `http://www.akamai.com/`.

[13] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr. 2000.

[15] L. Bent and G. Voelker, "Whole page performance," in *Workshop on Web Content Caching and Distribution (WCW)*, Boulder, CO, Aug. 2002.

[16] V. Paxson and M. Allman, "Computing TCP's retransmission timer," Nov. 2000, Internet RFC 2988.

[17] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle, "Stochastic models for generating synthetic HTTP source traffic," in *Proceedings of IEEE INFOCOM '04*, Hong Kong, China, Mar. 2004.

[18] A. Medina, J. Padhye, and S. Floyd, "Measuring the evoluation of transport protocols in the Internet," Tech. Rep., 2004.

[19] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites," in *Proceedings of WWW '02*, Honolulu, HI, May 2002.

[20] M. Allman, E. Eddy, and S. Ostermann, "Estimating loss rates with TCP," 2003, work in Progress.

[21] A. Habib and J. Chuang, "A measurement-based analysis of residential multihoming," in *IEEE INFOCOM '05, poster session*, Miami, FL, Mar. 2005.

[22] U. Fiedler, "Provisioning networks for web services," Dec. 2001. [Online]. Available: `http://www.tik.ee.ethz.ch/~fiedler/provisioning.html`

[23] P. Natarajan, J. Iyengar, P. Amer, and R. Stewart, "SCTP: an innovative transport layer protocol for the web," in *Proceedings of WWW '06*, Edinburgh, Scotland, May 2006.

[24] M. Christiansen, K. Jeffay, D. Ott, and F. Smith, "Tuning RED for web traffic," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 249–264, 2001.

[25] "The Apache Software Foundation," [Online]. Available: `http://www.apache.org/`.

[26] V. Padmanabhan and J. Mogul, "Improving HTTP latency," *Computer Networks and ISDN Systems*, vol. 28, no. 1–2, pp. 25–35, Dec. 1995.

[27] P. Barford and M. Crovella, "A performance evaluation of hyper text transfer protocols," in *Proceedings of ACM SIGMETRICS '99*, Atlanta, GA, May 1999.

[28] H. Kruse, M. Allman, and P. Mallasch, "Network and user-percieved performance of web page retrivals," in *Conference on Telecommunications and Electronic Commerce*, Neshville, TN, Nov. 1998.

[29] J. Heidemann, "Performance interactions between P-HTTP and TCP implementations," *ACM Computer Comm. Review*, vol. 27, no. 2, pp. 65–73, Apr. 1997.

[30] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP behavior of a busy Internet server: Analysis and improvements," in *Proceedings of IEEE INFOCOM '98*, San Francisco, CA, Mar. 1998.

[31] D. Lu, Y. Qiao, P. Dinda, and F. Bustamante, "Modeling and taming parallel TCP on the wide area network," in *Proceedings of IPDPS '05*, Denver, CO, Apr. 2004.

[32] T. Hacker, B. Noble, and B. Athey, "The effects of systemic packet loss on aggregate TCP flows," in *Supercomputing '02*, Los Alamitos, CA, Feb. 2002.

[33] J. Hall, I. Pratt, and A. Moore, "The effect of early packet loss on web page download times," in *Proceedings of Passive and Active Measurement (PAM) Workshop*, La Jolla, CA, Apr. 2003.

[34] D. Bernstein, "SYN cookies," 1996, [Online]. Available: `http://cr.yp.to/syncookies.html`.

[35] D. Anderson, H. Balakrishnan, M. Kaashoek, and R. Rao, "Improving web availability for clients with MONET," in *Proceedings of NSDI '05*, Boston, MA, May 2005.

[36] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, Sept. 1999.