In *Proceedings of the 2000 ACM SIGCOMM Conference*
September 2000

Critical Path Analysis of TCP Transactions

Paul Barford and Mark Crovella
Computer Science Department
Boston University
111 Cummington Street
Boston, MA 02215
barford,crovella@cs.bu.edu

ABSTRACT

Improving the performance of data transfers in the Internet (such as Web transfers) requires a detailed understanding of when and how delays are introduced. Unfortunately, the complexity of data transfers like those using HTTP is great enough that identifying the precise causes of delays is difficult. In this paper we describe a method for pinpointing where delays are introduced into applications like HTTP by using *critical path analysis.* By constructing and profiling the critical path, it is possible to determine what fraction of total transfer latency is due to packet propagation, network variation (*e.g.,* queuing at routers or route fluctuation), packet losses, and delays at the server and at the client. We have implemented our technique in a tool called `tcpeval` that automates critical path analysis for Web transactions. We show that our analysis method is robust enough to analyze traces taken for two different TCP implementations (Linux and FreeBSD). To demonstrate the utility of our approach, we present the results of critical path analysis for a set of Web transactions taken over 14 days under a variety of server and network conditions. The results show that critical path analysis can shed considerable light on the causes of delays in Web transfers, and can expose subtleties in the behavior of the entire end-to-end system.

1. INTRODUCTION

Response time is one of the principal concerns of users in the Internet. However, the root causes of delays in many Internet applications can be hard to pin down. Even in relatively simple settings (such as the retrieval of a single, static Web page) the complexities are great enough to allow room for inconclusive finger-pointing when delays arise. All too often, network providers are blaming the servers at the same time server managers are blaming the network.

Considerable effort has gone into improving the performance of servers and networks. Research into server design has led to many advances, including highly scalable hardware configurations [4, 9] and highly optimized server implementations such as [14]. Focus on the network has led to improvements in protocols such as the 1.1 version of HTTP [15]. However, to date, much of the research aimed on improving Internet application performance has focused on either the server or the network in isolation. While all of these improvements are valuable, future enhancements will most likely require a more detailed understanding of the interaction between networks and end systems.

In this paper we focus on methods that aid in pinpointing the sources of delays in Internet applications—allowing conclusive, accurate assignment of delays to either the network or the server. Furthermore, our methods are aimed at studying the network and the server together, as a complete end-to-end system. We focus in particular on Web transactions, looking to understand the relative and joint contribution that servers and networks make to overall transfer latency. Our general approach is to study these transactions using the method of *critical path analysis* [25].

Critical path analysis is the natural tool for understanding distributed applications because it identifies the precise set of activities that determine an application's performance [17, 27, 41]. The central observation of critical path analysis as applied to distributed systems is that only *some* of the component activities in a distributed application are responsible for the overall response time; many other activities may occur in parallel, so that their executions overlap each other, and as a result do not affect overall response time. In other words, reducing the execution time of any of the activities on the critical path will certainly reduce overall response time, while this is not necessarily true for activities off the critical path.

Recent work in [22] indicates that HTTP/1.0 is the dominant protocol used in the World Wide Web today. In this paper we apply critical path analysis to HTTP/1.0 transactions that use TCP Reno (as specified by RFC 2001 [39]) for their transport service.[1] We show how to do critical path analysis for TCP Reno, and describe how we use the resulting critical path to determine the fractions of response time

Supported in part by NSF Grant CCR-9706685 and by the Xerox Foundation.

[1]While our implementation is specific to TCP Reno, the methods discussed could easily be extended to encompass the packet loss recovery mechanisms of other versions of TCP such as New-Reno or SACK.

that are due to server delays, client delays, and network delays (including packet loss delay, propagation delay, and delay due to network variations such as queuing at routers, route fluctuations, etc.). The analysis has the appealing property that improvements in any of the delays we identify would surely have improved response time. This suggests that looking at the fraction of response time assigned to each category can provide a guide to which part of the system is most profitably improved. Despite the strengths of our approach, there are some inherent limitations which are described in Section 3.6.

Properly constructing the critical path depends on the details of the TCP implementation being used. However we have been able to develop a method that we believe applies to any TCP Reno implementation compliant with RFC 2001 for slow start, congestion avoidance, fast retransmit and fast recovery. Our approach does not require any instrumentation of end systems; we only require passively collected network traffic traces to be taken at the end points during the transaction. In this paper, we describe the method in detail and show its implementation in a tool called `tcpeval`. `tcpeval` takes as input endpoint traces from `tcpdump` along with implementation-specific constants and, by tracking the evolution of TCP's state as it controls packet flow, determines the critical path for a TCP flow. We then *profile* this critical path, assigning delays to various categories.

We have used `tcpeval` in a pilot study to demonstrate its utility. We show the results of critical path analysis for Web transfers using both Linux and FreeBSD TCP implementations over the wide area Internet. The results show that critical path analysis can give considerable insight to the root causes of delays in Web transfers. In particular, we find that for the systems we have measured, server load is the major determiner of transfer time for small files, while network load is the major determiner for large files; and we are able to characterize the points in the HTTP transaction where server delays are most often introduced. We also show that critical path analysis can pinpoint sources of variability in Web transfers: while packet losses are the primary contributor to variability of transfer time, we also find that the number of packets on the critical path can show very high variability.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 describes the process of constructing the critical path for TCP transactions in detail; and Section 4 describes the results of applying critical path analysis to data taken from both Linux and FreeBSD. Finally, Section 5 summarizes and concludes.

2. RELATED WORK

Previous work in the performance analysis and improvement of Web transactions falls into two categories: work on servers, and work on networks/protocols.

Work on servers has emphasized techniques for improving their performance, for example [2, 6, 8]. Such studies show how Web servers behave under a range of loads (including overload conditions) and have suggested enhancements to application implementations and the operating systems on which the servers run. However, unlike this paper, such studies have not to date considered how interactions with networks and clients in the wide area affects performance.

On the networking side, considerable previous work has focused on improving performance of the network infrastructure for Internet applications. Studies of network dynamics [10, 34] and improvements to the transport layer [11, 21, 32] have focused on application-independent issues. There has been some work studying TCP behavior within the context of Web transactions [5, 16, 29, 30]; such studies have resulted in a variety of enhancements to HTTP including data compression, persistent connections and pipelining which are all part of the HTTP/1.1 specification [15]. However none of these studies looks at how end system behavior interacts with network and protocol performance.

Recently, analytic models for steady-state TCP throughput and latency have been developed in [12, 26, 31]. However such models assume that no delays are introduced in the end systems; our approach specifically attempts to assess the effects of such delays on overall response time.

Our work depends on packet trace analysis, and extends the set of tools available to work with packet traces. The most basic problem with packet trace analysis is that traces are usually large and complex, so it can be difficult to identify key information in a trace. The most simple analysis techniques are graphical such as time line plots or sequence plots [21, 38] which enable one to understand the data and acknowledgment sequences of a transaction. In [11], Brakmo *et al.* developed a sophisticated graphical tool which enables multiple details of a TCP transaction to be followed simultaneously. In [33], Paxson describes the tool `tcpanaly` which he used to document a wide variety of unusual behaviors from a number of different TCP implementations. Paxson's work is also significant for our study because it points out the pitfalls of packet traces which we had to address in the development of `tcpeval`. While we have found no references to "critical path" analysis of TCP, in [35] Paxson discusses the notion of "cause-effect" issues in TCP transactions, which is an idea that also underlies our approach. However, Paxson had difficulty fully developing this notion because he was interested in establishing this relationship by measuring at a single point along the end-to-end path. Finally, while they do not explicitly use the term "critical path", Schroeder and Burrows essentially do critical path analysis of the Firefly RPC mechanism in [37]. In this work the authors define a static set of events which make up the *fast path*. They either estimate or measure the performance of each event in a local area environment. They then suggest a series of improvements to the fast path which they estimate can speed up the RPC mechanism by a factor of three.

Broadly, our work provides significant insight to the question: "Why is my Web transfer so slow? Is it the network or the server?" This question is at the heart of a number of other studies and commercial products. The Keynote system [19] measures response time by timing overall delay from the perspective of the client; as a result it has a hard time accurately differentiating between server and network delays. A more accurate approach is used in net.Medic [20] which infers server delays from packet-level measurements. More recently, Huitema [18] presents measurements that differentiate between network and server delay by equating server delay with the time between HTTP GET and receipt of the first data packet. Results we present in Section 5 support this idea as a good first-order approximation — we show that in many cases (*e.g.,* for small transfers) the majority of server-induced delay occurs at precisely this point in the transaction. However, for large transfers when the server is under heavy load, we also show that there can be significant server delays long after connection start up. The work in [18] also points out the fact that DNS lookups can also be a significant cause of delay; since we are mainly interested in network/server interaction, we do not consider these sources of delays in this paper.

3. MECHANICS OF CRITICAL PATH ANALYSIS

In this section we describe how to discover the critical path for TCP flows; details of the algorithm we use; how critical path discovery can be employed in an HTTP setting; and how we profile the critical path to accurately assess the causes of transfer delays. We also describe the tool `tcpeval` which performs these analyses.

3.1 Discovering the Critical Path for a Unidirectional TCP Flow

We start by restricting our attention to the heart of our approach, the analysis of the data transfer portion of a unidirectional TCP flow; in the next section we will discuss bidirectional flows and application-level protocols. Critical path analysis of unidirectional flows is based on an examination of the *packet dependence graph* (PDG) for a particular TCP flow.

We define the packet dependence graph as a weighted, directed graph (in fact, a tree) in which each node represents the arrival or departure of a single packet at either endpoint. Thus, each packet that is not dropped en route corresponds to two nodes in the PDG. An illustration of the tree structure of the PDG's is shown in in Figure 1. This figure shows an idealization of the slow start portion of TCP. On the left is a traditional time line diagram where the dependencies between data and ACK packets are not explicit. On the right we draw the PDG where dependencies are explicit.

Arcs in the PDG represent a dependence relation corresponding to Lamport's *happened-before* relation [23]; each arc is weighted by the elapsed time between the events that form its start and end points. In this case, the dependence relation is that defined by the semantics of reliable data transport using TCP Reno. There are four kinds of arcs in the graph: first, for each distinct packet that is not dropped, we construct an arc from its departure at one endpoint to its arrival at the other endpoint. Second, for each acknowledgment packet (ACK), we find the data packet containing the last byte being acknowledged by the ACK. We construct an arc from the arrival of that data packet to the departure of that ACK packet, reflecting the fact that the ACK could not have been emitted before the corresponding data packet was received. The third type of arc is from ACK packets to data packets. For each ACK packet, we determine the data packets that it "liberated", and construct an arc from the arrival of the ACK to the departure of each liberated

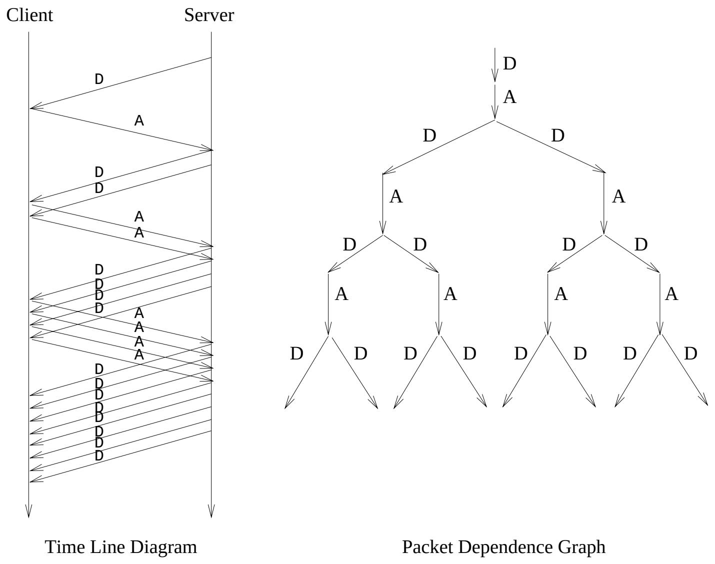


Figure 1: A time line diagram of a TCP flow and its reflection in a Packet Dependence Graph (PDG). The PDG shows which data packets (D) are liberated by individual ACKs (A).

data packet. By "liberating" a data packet, we mean that the arrival of the ACK caused TCP to open its congestion window sufficiently to allow the transmission of the data packet. Tracking the liberation of data packets is central to our approach, and must be done carefully; we discuss it in detail in Section 3.3.

In the cases where no packet loss occurs, these three arc types are sufficient. However, when a packet is lost, eventually the packet is retransmitted, and there is a dependence relation between the two events. In this case, we construct an arc from the departure of the dropped packet to the departure of the earliest subsequent retransmission.

Having constructed the weighted tree described by this process, we define the critical path in the usual way; that is, the longest path in the tree starting from the root. Since we are considering a unidirectional flow, the root is the first data packet sent. The result is a (weighted) chain of dependence relations running from the connection's first data packet to its last data packet sent.

3.2 Discovering the Critical Path for an HTTP Transaction

The discussion so far has assumed that there are no application level dependencies among events. This is not the case in general; for example, it may be that the departure of data packets sent in one direction may depend on the arrival of data packets traveling in the other direction, due to application-level semantics. In this study we focus on HTTP/1.0 retrievals of static files (without persistent connections), for which this dependence only exists between the last packet comprising the HTTP GET and the first data packet sent in response. Nevertheless, to properly construct the critical path, this dependence must be taken into account. Nothing in our methods precludes the analy-

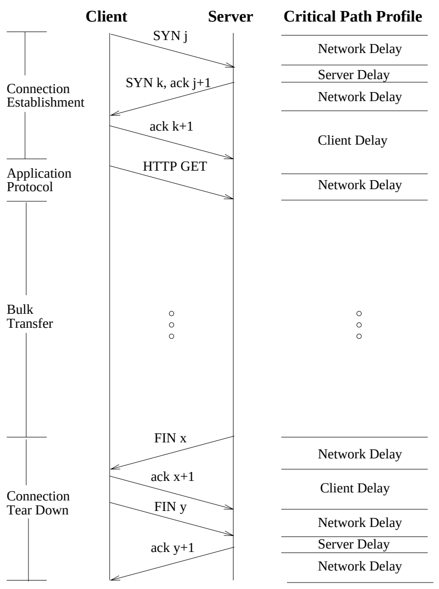


Figure 2: Critical path analysis for TCP connection setup, HTTP GET and TCP connection tear down.

sis of HTTP/1.1 however our implementation is specific to HTTP/1.0.

Since we do not want to restrict the applicability of our methods to any particular application, we do not attempt to explicitly incorporate application-level dependence into the PDG. Instead, we construct the complete critical path by breaking the entire HTTP transaction into four parts: 1) connection establishment — the SYN, SYN-ACK, ACK sequence; 2) the packet sequence delivering the HTTP GET and associated parameters; 3) the returning bulk data transfer; and 4) connection tear-down — the FIN, ACK, FIN, ACK sequence. This breakdown is shown in Figure 2 (details of the bulk transfer portion of the critical path are discussed in Section 3.3).

The advantage of this decomposition is that the parts do not overlap; the dependence between each part is clear; and that parts 1, 2, and 4 do not depend on the size of the file, and so can be easily analyzed with special-purpose code. Each of parts 1, 2, and 4 follows one of a small set of patterns, and the patterns only vary when packets are lost, which is easily accounted for. Thus, discovering the critical path for HTTP/1.0 transactions requires relatively straightforward analysis of the connection setup, file request, and connection tear down sequences, combined with the unidirectional-flow algorithm described in the previous section for bulk data transfer.

3.3 Implementing Critical Path Discovery for Bulk Data Transport in TCP Reno

The critical path discovery algorithm for TCP Reno described in Section 3.1 can be implemented in two stages: a forward pass over the packet trace and a backward pass up the critical path.

The purpose of the forward pass is to construct a *round* for each ACK. Each round consists of the ACK and the data bytes which are liberated by that ACK. The data bytes liberated by a specific ACK are determined by the state of the receive and congestion windows on the server when the ACK was received. We identify the data bytes liberated by each ACK by simulating the receive and congestion window states based using the TCP Reno slow start/congestion avoidance/fast retransmit/fast recovery algorithms as specified in RFC 2001 [39] and described in [40]. The reason for this accounting is that an ACK may be received at the server before it has had a chance to transmit all of the packets permitted by the current window state; thus it is not safe to assume that each data packet was liberated by the most recent arriving ACK. After establishing the bytes liberated by an ACK, we determine the data packet that triggered the ACK (by comparing sequence numbers) and maintain a pointer to that data packet with the round for the ACK. The result is a tree of rounds.

After the forward-pass construction of all rounds, the critical path is determined by tracing back up the tree, starting from the last data packet (before the FIN is sent, or containing the FIN). At each round, we accumulate arcs onto the critical path, and then move to the parent round. This continues until reaching the first round, which will contain the first data packet.

An example bulk transfer for a Linux TCP stack is shown in Figure 3. Key steps in critical path analysis are left-to-right: capture of packet traces, analysis of rounds, construction of critical path, and profiling of the critical path. In this figure, rounds 1, 2, and 3 progressively consist of two, three, and four data packets, since the connection is in slow start [21]. The ACK for round 4 only acknowledges two of the four packets transmitted in round 3. Thus, round 4 is responsible for the release of only three more data packets—since from the server's perspective, there are still two data packets in flight and the current size of the congestion window is five—which illustrates the need to maintain the correct receiver and congestion window state to properly construct dependence arcs.

Maintaining the proper receiver and congestion window size in our simulation is complicated when packets are dropped, since the retransmissions signaled by coarse-grained timeouts and fast retransmits call for different congestion window size adjustments. Fortunately, we can distinguish between drops that are recognized by coarse-grained time outs and those which are recognized by the fast retransmit mechanism as follows: all retransmissions are visible (since we have traces from the sender's endpoint) and all duplicate ACK's are also visible; thus when a packet is retransmitted, we know whether it was in response to a coarse-grained timeout or in response to the receipt of 3 duplicate ACKs. We note that this means it is not necessary to simulate TCP's retransmission timer to determine when coarse-grained timeouts occur, which would be very difficult to reconstruct after the fact, and would introduce a number of TCP implementation dependent issues. One potential issue is that there may be rare instances where coarse-grained timeouts are the actual trigger of a retransmitted packet even though a third duplicate ACK has been received. We call these instances *hidden timeouts*. Our mechanism does not currently distinguish hidden timeouts. If they were to occur, our TCP simulation would have larger congestion windows leading to obviously incorrect client delays which can be detected by our critical path analysis tool, `tcpeval`. In the analysis of our sample data, we found no instances of hidden timeouts.

The strength of the critical path as an analytic tool is also evident from the figure. Consider once again the data transfer shown in Figure 3. The last packet (before the FIN) was released by the ACK in round 7, which is therefore on the critical path. The data which triggered the ACK for round 7 was released by round 6, which is also on the critical path. The ACK in round 6 was triggered by a retransmitted packet, which was originally liberated by round 4. This shows that none of the packets in round 5 are on the critical path, and indeed, even considerable variations in the delivery times of packets in round 5 would have had no effect on the eventual completion time of the overall transfer.

3.4 Profiling the Critical Path

Once the critical path has been constructed, we can use it to identify the sources of delays in the data transfer. To do so we assign a natural category to each arc type; the critical path profile is then the total contribution to each category made by arc weights on the critical path. We map arcs to categories as follows: arcs between differently-typed nodes (data $\rightarrow$ ACK or ACK $\rightarrow$ data) on the same endpoint are assigned to that endpoint (*i.e.,* **server** or **client**); arcs between equivalently-typed nodes (data $\rightarrow$ data or ACK $\rightarrow$ ACK) on opposite endpoints are assigned to the **network**; and arcs between data nodes on the same endpoint are assigned to **packet loss**.[2] These correspond to the intuitive sources of delay in each case. Sample delay assignments can be seen at the far right side of Figure 3.

We can refine this classification somewhat. First, it is possible to distinguish between packet losses recognized by coarse-grained timeouts, and those signaled by the receipt of 3 duplicate ACKs from the receiver (fast retransmits). Second, each network delay can be decomposed into two parts: propagation delay and network variation delay. We define propagation delay to be the minimum observed delay in each direction over *all* experiments in which the path did not change. Delay due to network variation is defined as the difference between network delay and propagation delay. It can be caused by a number of things including queuing at the routers along the end-to-end path and route fluctuation.

The result is a six-part taxonomy of delays in HTTP transfers, measured along the critical path: 1) server delays, 2) client delays, 3) propagation delays, 4) network variation delays, 5) losses recognized by coarse-grained timeouts, and 6) losses signaled by the fast retransmit mechanism.

3.5 Realization of Critical Path Analysis in `tcpeval`

The critical path construction method described in Sections 3.1 and 3.3 is general in the sense that it only depends on accurate tracking of TCP receiver and congestion window size—so the method works for any implementation that is compliant with RFC 2001. This robustness is evident in Section 4 which shows results from two different TCP implementations (Linux 2.0.30 and FreeBSD 3.3). In addition, while we focus on HTTP/1.0 in this paper, other protocols could be evaluated, given accurate modeling of application-level packet dependences.

We have implemented the entire critical path analysis process (path construction and profiling) in the tool `tcpeval` consisting of approximately 4,000 lines of C code. In addition to critical path analysis, it provides statistics for higher level HTTP transaction study (such as file and packet transfer delay statistics) and produces time line and sequence plots for visual analysis.

The principal inputs to `tcpeval` are `tcpdump` traces taken at the client and the server. In addition to the packet traces, the only other inputs to `tcpeval` are the initial values for the congestion window (CWND) and the slow start threshold (SSTHRESH) since these can vary between TCP implementations.

A number of implementation issues arise due to our use of `tcpdump` traces and due to our need to accurately measure one-way packet transit times. These are discussed in the context of automated packet trace analysis by Paxson [33]. Specifically, these are the problems which can occur in packet filters such as the Linux Packet Filter (LPF) or the Berkeley Packet Filter (BPF) which are used by `tcpdump` to extract packet traces. First, packets can dropped by the filter. If `tcpeval` sees an ACK or other response for a packet which was supposedly lost, `tcpeval` concludes it was dropped in the filter rather than the network. Second, packets can be added to the trace by the filter. To address this, if `tcpeval` sees the same packet twice and it was not a retransmit, it simply discards the second copy of the packet. Third, packet filter can report packets in a different order than the occurred in reality. `tcpeval` insures the correct order by sorting packets by time stamp in a number of places during the analysis. Last, packet filters can incorrectly time stamp packets which causes a different type of reordering in the packet trace. At present, `tcpeval` does not account for this. The effect will be that packets will appear to be sent beyond the congestion window which is flagged as an error by `tcpeval`.

With respect to accurate measurements of one-way delays, we have carefully configured NTP [28] (using frequent hard resets and syncing with multiple servers where needed) in our experimental setup to obtain the closest possible clock synchronization. In this way we have found that for the two experimental configurations in Section 4, the resulting difference in time between nodes was on the order of 1 ms as reported by NTP. Nonetheless, the nature of our approach demands highly synchronized clocks, so we will configured our systems with GPS-synchronized clocks (enabling synchronization on the order of microseconds) for future experiments.

[2]Some server delay is included in what we call delay due to packet loss since upon either coarse-grained timeout or receipt of a third duplicate ACK, a heavily loaded server could take some time to generate a retransmitted packet.

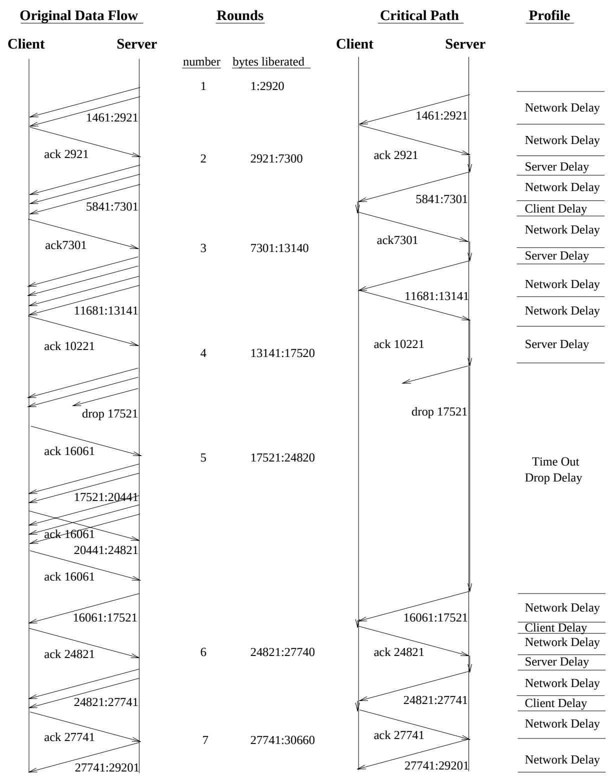


Figure 3: Details of critical path analysis for the bulk transfer portion of a unidirectional Linux TCP flow.

3.6 Limitations

There are a number of limitations to the use of critical path analysis in this setting. First, there are some details of the web transactions which we do not take into account. The most obvious is that we do not consider the delay due to domain name server (DNS) resolution. `tcpeval` easily could be enhanced to do this by tracking DNS packets. We also do not consider client browser parse times in our experiments since we simply transfer files as is explained in Section 4.

There are also some aspects of our application of CPA to TCP transactions that limit our ability to use the strengths of CPA. The most fundamental is the inability to do "what if" analysis after establishing the CPA for a TCP transaction. Although one could explore the effects of changing delays or drops on a particular critical path, in reality, changing delays or drop events can change the dependence structure of the entire transaction, (*i.e.,* the PDG). For example, when a drop occurs, it changes the congestion window size and thus the number of packets which are liberated by subsequent ACKs. This fact limits the extent of "what-if" analysis to *simulating* what would happen in the remainder of a transaction instead of actually being able to know exactly what would have happened.

4. CRITICAL PATH ANALYSIS APPLIED TO HTTP TRANSACTIONS

In this section we apply critical path analysis to measurements of HTTP transactions taken in the Internet.

4.1 Experimental Setup

The experiments presented here use a distributed infrastructure consisting of nodes at Boston University, University of Denver and Harvard University. At Boston University is a cluster consisting of a set of six PCs on a 100 Mbps switched Ethernet, connected to the Internet via a 10Mbps bridge. One of the PCs runs the Web server itself (Apache version 1.3 [36] running on either Linux v2.0.30 or FreeBSD v3.3) as well as operating system performance monitoring software. Two more PCs are used to generate local load using the SURGE Web workload generator [7], which generates HTTP requests to the server that follow empirically measured properties of Web workloads. The requests generated by SURGE are used only to create background load on the server. The fourth PC collects TCP packet traces using `tcpdump`, the fifth makes active measurements of network conditions (route, propagation delay and loss measurements) [3] during tests and the last system is used to manage the tests.

Off-site locations hold client systems which generate the monitored requests to the server. Client 1 (running Linux v2.0.30) was located at University of Denver, 20 hops from the server cluster at Boston University; the path to that site includes two commercial backbone providers (AlterNet and Qwest). Client 2 (running FreeBSD v3.3) was located at Harvard University, 8 hops from the server cluster at Boston University. The path to the Harvard client travels over the

[3]Route measurements were taken in each direction at five minute intervals during tests. Only one instance of a route change during a test was observed.

vBNS.

In our experiments we explore variations in network load conditions, server load conditions, file size, and network path length. To explore a range of different network conditions, we performed experiments during busy daytime hours as well as relatively less busy nighttime hours. To study the effects of server load, we used the local load generators to generate either light or heavy load on the server during tests. We used settings similar to those explored in [8] to place servers under light load (40 SURGE user equivalents) or heavy load (520 SURGE user equivalents). The work in [8] shows that 520 user equivalents places systems like ours in near-overload conditions. The local load generators requested files from a file set of 2000 distinct file ranging in size from 80 bytes to 3.2MB. Main memory on the server is large enough to cache all of the files in the file set so after the initial request (which is fetched from disk), all subsequent requests for a file are served from main memory.

To explore variations in file size, the monitored transactions (generated at the remote clients) were restricted to three files of size 1KB, 20KB, or 500KB. In choosing these sizes, we were guided by empirical measurements [3, 13]. The 1KB file was selected to be representative of the most common transmissions, small files that can fit within a single TCP packet. The 20KB file was selected to be representative of the median sized file transferred in the Web. The 500KB file was selected to be representative of large files, which are much less common but account for the majority of the bytes transferred in the Web.

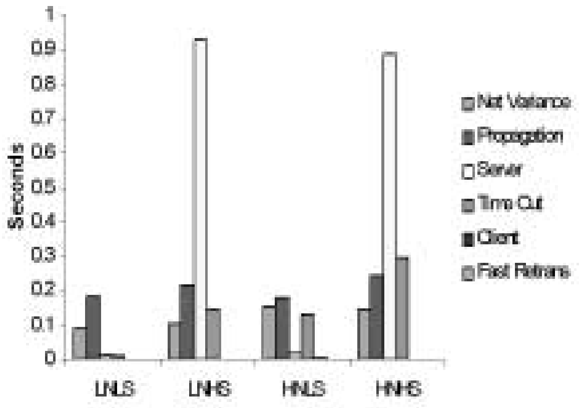
The combinations of network load (light or heavy), server load (light or heavy) and downloaded file size result in 12 separate tests per day. Each test consisted of downloading files of a given size for one hour. Traces were gathered over 10 weekdays from Client 1 (Linux, long path) and 4 days from Client 2 (FreeBSD, short path). For each client, we profiled the critical path of each HTTP transaction, and within a given test type (file size, network load, network path, and server load) we averaged the resulting critical paths for analysis.

4.2 Results

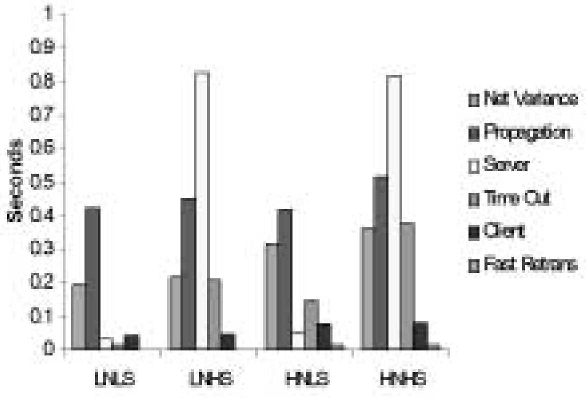
We first examine the resulting critical path profiles for transfers to Client 1. These results are shown in Figure 4. Figure 4(a), (b), and (c) show small (1KB), medium (20KB), and large (500KB) file sizes respectively; within each chart the particular experiments are denoted by H or L for high or low load and N or S for network and server.

A number of observations are immediately evident from the figure. At the highest level, it is clear that the category that is the most important contributor to transaction delay depends on a number of factors, including file size and server load. This shows that no single aspect of the system is generally to blame for long transfer times, and that the answer is rather more complicated.

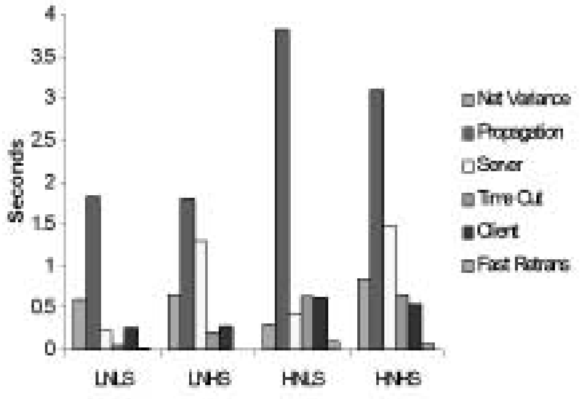
In addition, the figure shows that propagation delay is often the most important overall contributor to transaction delay (*e.g.,* for large files, and for all file sizes when server load is low). This is an encouraging statement about the de-



(a)



(b)



(c)

Figure 4: Aggregate mean contributions to transfer delay from Client 1 for (a) small file, (b) medium file and (c) large file. LNLS = low network load, low server load, HNHS = high network load, high server load.

sign of the end-to-end system, since this component of the data transport process is rather hard to avoid. When propagation delay dominates transfer time it would seem that only by opening the congestion window more aggressively could overall end-to-end response time be significantly reduced (while recognizing the difficulty of doing so without contributing to congestion) [32].

Overall, we note that when server load is low, the delays on the client side and the server side are roughly comparable. Client delays throughout tend to be quite low; since the remote site consists of a PC simply transferring a single file, this is consistent with expectations. In addition, this data confirms that there are many more retransmissions triggered by time-outs than by the fast retransmit mechanism in all our cases. This situation has been observed in other studies [5, 24, 31].

4.2.1 Effect of File Size.

Comparing critical path profiles across file sizes, we can make the following observations from Figure 4.

Small files are dramatically affected by load on the server. When server load is low, network delays (network variation and propagation) dominate. However, when server load is high, server delays can be responsible for more than 80% of overall response time. Since small files are so common in the Web, this suggests that performance improvements quite noticeable to end users should be possible by improving servers' delivery of small files. For *medium sized files,* network delays also dominate when server load is low; but when server load is high, the contribution to delay from the network (network variation and propagation) and the server are comparable. So for these files, neither component is principally to blame. Finally, for *large files,* delays due to the network dominate transfer time, regardless of server load.

4.2.2 Effect of Server Load.

By examining individual critical paths, we can observe that the delays due to server load are not spread uniformly throughout the transfer. In fact, for small and medium files, nearly all of the server delay measured in our HTTP transactions occurs between the receipt of the HTTP GET at the server and the generation of the first data packet from the server. In Figure 5 we plot some typical critical paths for small and medium files, comparing the cases of low and high server load (network load is high throughout).

Each diagram in the figure shows the critical path for the transfer of a single file, with time progressing downward from the top, and lines representing dependence arcs in the PDG. In each diagram, the client is on the left (sending the first packet, a SYN) and the server is on the right. The figure shows that when server load is high, there is a characteristic delay introduced between the moment when the HTTP GET arrives at the server (which is in the second packet from the client, because the second handshake ACK is piggybacked), and when the first data packet is sent out in response. We note that this server start-up delay may be specific to the Apache 1.3.0 architecture; we are currently investigating the case for other servers and versions of Apache.

Although these figures clearly show that substantial server

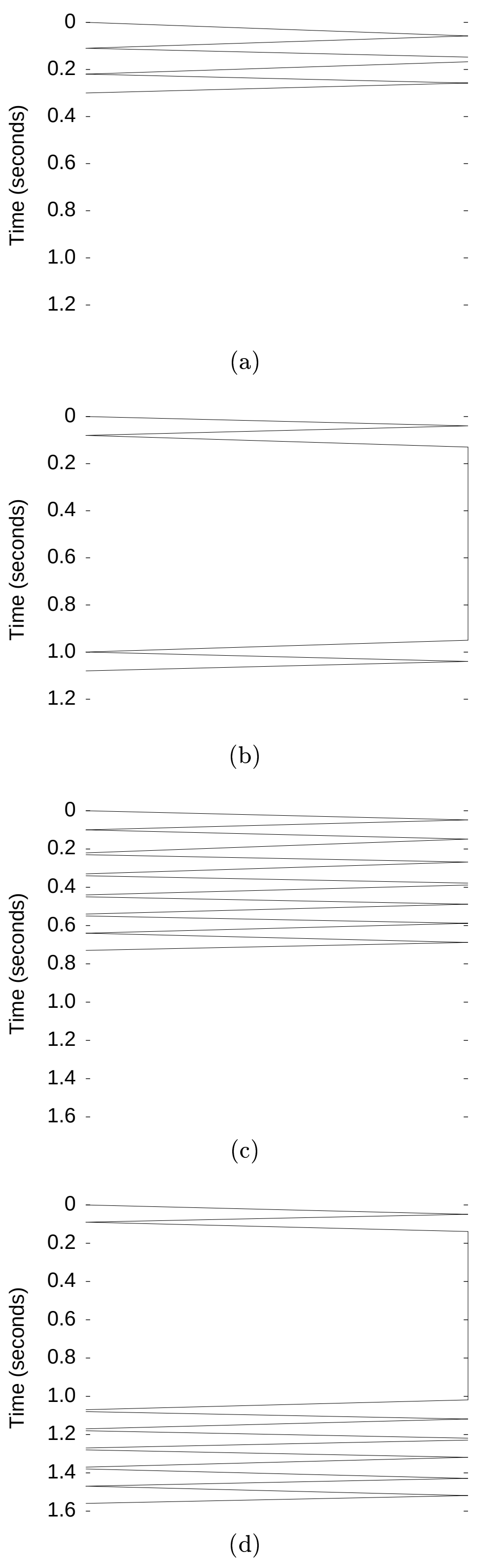
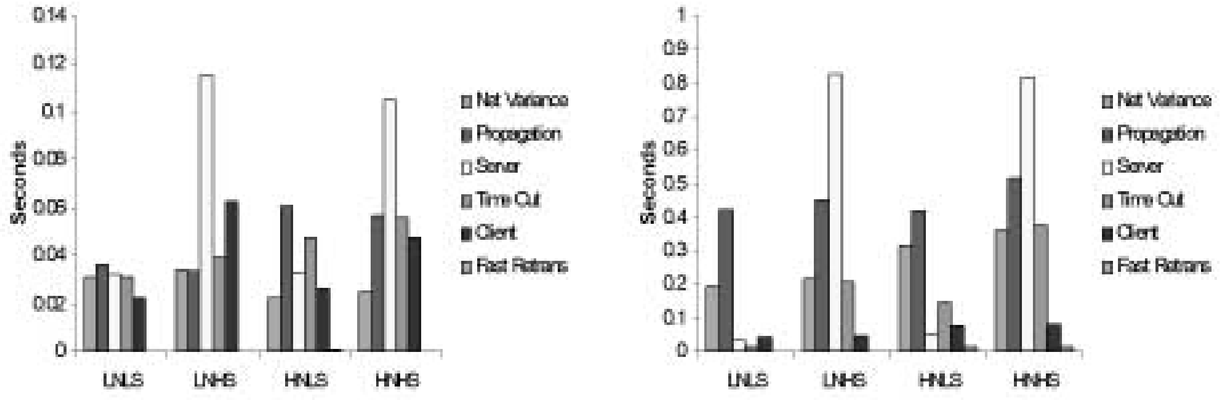


Figure 5: Critical path diagrams for the transfer of (a) small file, light server load, (b) small file, heavy server load, (c) medium file, light server load, (d) medium file, heavy server load.



(a) (b)

Figure 6: Aggregate mean contributions to transfer delay (a) from Client 2 (short path, FreeBSD) and (b) from Client 1 (long path, Linux) for medium sized file.

delay is introduced at connection start-up, we can also determine that for large files, a busy server introduces considerable delays later in the transfer as well. This can be seen from Figure 4 comparing (a) or (b) with (c); the total server-induced delay is twice as large in (c), although the startup delay is independent of file size.

Taken together, these results on the file size and server load indicate that while both servers and networks show startup costs at the beginning of transfers, the startup cost on our server (when under load) is much higher. On the other hand, when approaching steady-state (*i.e.,* during long transfers) the network introduces more delays than does the server.

4.2.3 *Effect of Network Load.*

Propagation delay as we have defined it is proportional to number of round-trips on the critical path. For small files, there are 6 packets on the critical path; for medium files, there are typically 14 packets on the critical path; and for large files, there are typically close to 56 packets on the critical path (all of these values are in the absence of losses). One-way delay in both directions between our site and Client 1 was determined (using minimum filtering) to be about 32ms.

Figure 4 shows that for small and medium sized files, propagation delay is independent of network load or server load, which agrees with intuition. However, for large files, propagation delays are higher under high network load. On first glance one might assume this is because of a difference in routes used during tests. This however is not the case. Upon detailed examination of the critical paths, it is apparent that there are actually more round-trips on the critical path on average in the cases of heavy network load. This is because TCP reduces its congestion window dramatically after coarse-grained timeouts. If coarse-grained timeout drops are rare, the effect will be that the congestion window will have time to grow large between drops. If however coarse-grained timeouts are frequent, the congestion window has less chance to open fully, and so more round-trips appear on the critical path.

In contrast to propagation delay, delay due to network variation generally increases when the network is heavily loaded. However, network variation delay overall is a less important contributor to transfer delay than is propagation delay; and between network variation delay and packet loss (both effects of congestion) neither is generally more significant than the other.

4.2.4 *Effect of Path Length.*

To study the effect of path length on the critical path profile, we ran experiments to Client 2. Results for medium sized files only are shown in Figure 6; on the left of the Figure we show results for Client 2; on the right of the Figure we repeat the results for Client 1 (from Figure 4) for comparison purposes.

The figure shows that file transfers occurred almost an order of magnitude faster over the much-shorter path to Client 2. In general this increases the importance of server delays to overall transfer time; for example, for the LNLS case, transfers to Client 2 were affected equally by propagation delay and by server delay, while the same transfers to client 1 were dominated by propagation delays.

Interestingly, we find that server delays are also radically reduced in the Client 2 setting; this may be due to the different systems software used. We are currently investigating the cause of this effect.

4.2.5 *Causes of Variability in Transfer Duration.*

Finally, while Internet users are typically interested in mean transfer duration, variability in transaction duration is also a source of frustration. Our results so far have presented only mean statistics; however by studying the variability of each category in our critical path profile we can also shed light on the root causes of variability in transfer duration. Figure 7 plots the standard deviation of each category of delay, for all three file sizes transferring to Client 1 (*i.e.,* this figure is analogous to Figure 4).

This figure exposes a number of important aspects of transfer time variability. First, coarse-grained timeout losses are the overwhelming contributor to the variability of transfer delay for small and medium sized files. This is because the duration of even a single timeout is often much larger than the typical total transfer time in the absence of timeouts for these file sizes; thus when coarse-grained timeouts occur, they are very noticeable.

A surprising result is that for large files, when the network is heavily loaded, a significant cause of transfer time variability is propagation delay. This means that the number of packets on the critical path is showing high variability under heavily loaded network conditions. This can be confirmed by examining the distribution of critical path lengths for large files, comparing conditions of low network load with high network load. These histograms (along with each distribution's minimum, mode, and mean) are shown in Figure 8.

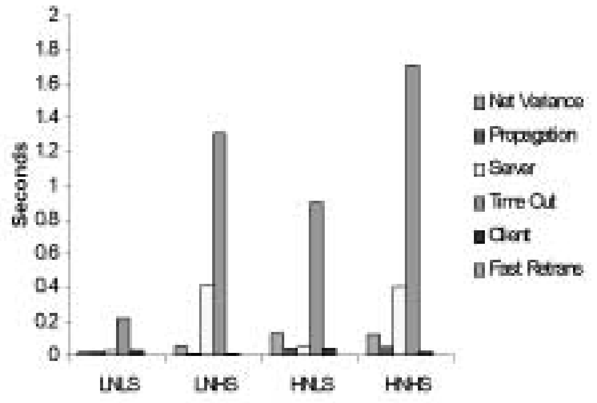
This Figure helps explain why mean propagation delay under high network load is larger (as discussed above); the summary statistics show that the common cases (the distributional modes) are the same, but that under heavy network load the critical path length distribution shows much longer tails.

5. CONCLUSIONS

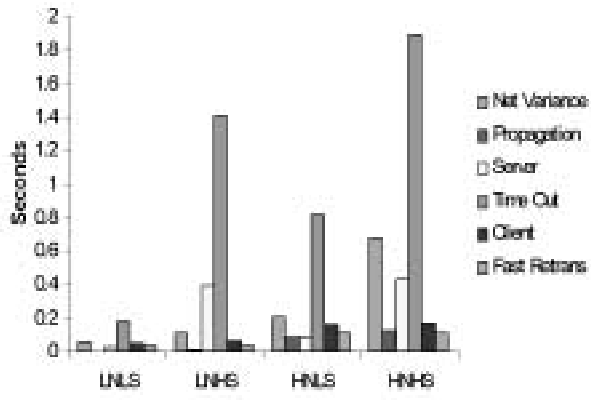
In this paper we have described methods and a tool for critical path analysis of Internet flows. We have shown how to construct the critical path for unidirectional TCP flows, and how to extend this method to HTTP transactions. Furthermore, we have developed a taxonomy of kinds of delays that can occur in TCP flows and used this taxonomy to profile the critical path.

An important goal of traffic analysis tools is wide applicability, and we have shown that the method we use for constructing the critical path applies to a wide range of TCP implementations, *i.e.,* those that conform to TCP Reno. Our tool does not require any intrusive instrumentation of end systems, only passively collected packet traces from both endpoints.

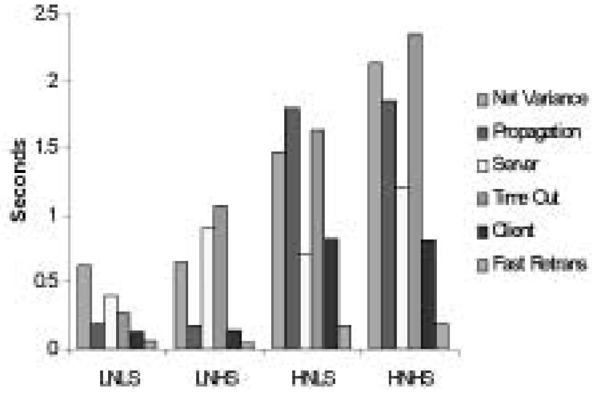
Applying this method to HTTP transfers in the Internet helps answer questions like "What are the causes of long Web response times?" We have shown that for the systems we have measured, server load is the major determiner of transfer time for small files, while network load is the major determiner for large files. If found to be general, this would lend support to the development of server enhancements to speed the transfer of small files. Furthermore, critical path analysis uncovers a number of more subtle points. For example, the contribution of propagation delay to the critical path is often greater than that network variability (*e.g.,* queuing); this suggests that it may be difficult to improve transfer latency in some cases without more aggressive window opening schemes such as those discussed in [1, 32] or that better retransmission algorithms which avoid timeouts and slow start [11]. In addition, the dominant cause of variability in transfer time is packet loss; but surprisingly, even



(a)

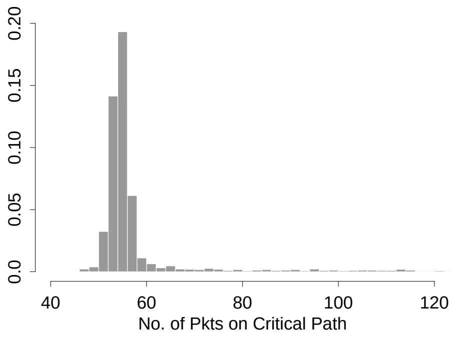


(b)

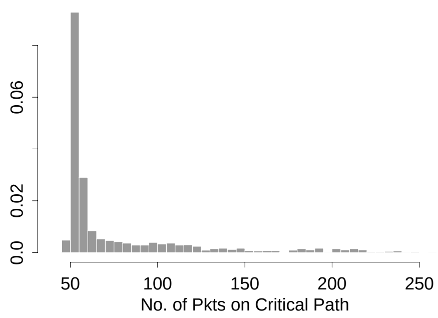


(c)

Figure 7: Standard deviations of delay components for (a) small file, (b) medium file and (c) large file.



(a: min: 47, mode: 55; mean: 58)



(b: min: 48 mode: 54; mean: 76)

Figure 8: Histograms of Number of Packets on Critical Path for large file: (a) Low network load (b) High network load (both are for Client 1, high server load).

the number of packets on the critical path for a constant-sized file can show a very long distributional tail.

6. ACKNOWLEDGEMENTS

The authors would like to thank Vern Paxson and Anja Feldmann for their input on various aspects of this work. The authors would like to thank Lars Kellogg-Stedman for his assistance with experimental systems used in this work. Many thanks also to David Martin and Peg Shaffer for hosting client systems used to collect data in this study. Thanks also to the anonymous reviewers of this paper.

7. REFERENCES

[1] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. IETF RFC 2414, September 1998.

[2] J. Almeida, V. Almeida, and D. Yates. Measuring the behavior of a world wide web server. In *Proceedings of the Seventh IFIP Conference on High Performance Networking (HPN)*, White Plains, NY, April 1997.

[3] M. Arlitt and C. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[4] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Conference*, San Diego, CA, June 2000.

[5] 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, March 1998.

[6] G. Banga and J. Mogul. Scalable kernel performance for internet servers under realistic loads. In *Proceedings of the USENIX Annual Technical Conference*, New Orleans, LA, June 1998.

[7] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, Madison, WI, June 1998.

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

[9] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed packet rewriting and its application to scalable server architectures. In *Proceedings of the 1998 International Conference on Network Protocols (ICNP '98)*, October 1998.

[10] J. Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of ACM SIGCOMM '93*, San Francisco, September 1993.

[11] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGMETRICS '96*, Philadelphia, PA, May 1996.

[12] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proceedings of the 2000 IEEE Infocom Conference*, Tel-Aviv, Israel, March 2000.

[13] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[14] P. Druschel, V. Pai, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, Monterey, CA, June 1999.

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. IETF RFC 2068, January 1997.

[16] H. Frystyk-Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Wium-Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1 and PNG. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, Setpember 1997.

[17] Jeffrey K. Hollingsworth and Barton P. Miller. Parallel program performance metrics: A comparison

and validation. In *Proceedings of Supercomputing '92*, November 1992.

[18] C. Huitema. Internet quality of service assessment. ftp://ftp.telcordia.com/pub/huitema/stats/quality_today.html.

[19] Keynote Systems Inc. http://www.keynote.com, 1998.

[20] Lucent NetCare Inc. net.medic. http://www.ins.com/software/medic/datasheet/index.asp, 1997.

[21] V. Jacobson. Congestion avoidance and control. In *In Proceedings of ACM SIGCOMM '88*, pages 314–332, August 1988.

[22] B. Krishnamurthy and C. Willis. Analyzing factors that influence end-to-end web performance. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Netherlands, May 2000.

[23] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[24] D. Lin and H.T. Kung. TCP fast recovery strategies: Analysis and improvements. In *Proceedings of IEEE INFOCOM '98*, San Francisco, CA, March 1998.

[25] K. G. Lockyer. *Introduction to Critical Path Analysis*. Pitman Publishing Co., New York, N.Y., 1964.

[26] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[27] Barton P. Miller, Morgan Clark, Jeff Hollingsworth, Steven Kierstead, Sek-See Lim, and Timothy Torzewski. IPS-2: The second generation of a parallel program measurement system. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):206–217, April 1990.

[28] D. Mills. Network time protocol (version 3): Specification, implementation and analysis. Technical Report RFC 1305, Network Information Center, SRI International, Menlo Park, CA, 1992.

[29] J. Mogul. The case for persistent-connection HTTP. Technical Report WRL 95/4, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[30] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, Setpember 1997.

[31] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, Setpember 1998.

[32] V. Padmanabhan and R. Katz. TCP fast start: A technique for speeding up web transfers. In *Proceedings of the IEEE GLOBECOM '98*, November 1998.

[33] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[34] V. Paxson. End-to-end internet packet dynamics. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[35] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California Berkeley, 1997.

[36] Apache HTTP Server Project. http://www.apache.org, 1998.

[37] M. Schroeder and M. Burrows. Performance of firefly rpc. In *Proceedings of the Twelfth ACM Symposium on Operating System Principles*, Litchfield Park, AZ, December 1989.

[38] T. Shepard. TCP packet trace analysis. Master's thesis, Massachusetts Institute of Technology, 1990.

[39] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, January 1997.

[40] W.R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[41] C.-Q. Yang and B. Miller. Critical path analysis for the execution of parallel and distributed programs. In *Proceedings of 8th International Conference on Distributed Computing Systems*, San Jose, CA, June 1997.