This paper is a draft that will appear in IEEE/ACM Transactions on Networking.
Final editing is still expected. Please replace it with the final version when published.

# Modeling the Performance of HTTP Over Several Transport Protocols*

John Heidemann Katia Obraczka Joe Touch

DRAFT June 6, 1997

## Abstract

This paper considers the interaction of HTTP with several transport protocols, including TCP, Transaction TCP, a UDP-based request–response protocol, and HTTP with persistent TCP connections. We present an analytic model for each of these protocols and use that model to evaluate network overhead carrying HTTP traffic across a variety of network characteristics. This model includes an analysis of the transient effects of TCP slow-start. We validate this model by comparing it to network packet traces measured with two protocols (HTTP and persistent HTTP) over local and wide-area networks. We show that the model is accurate within 5% of measured performance for wide-area networks, but can underestimate latency when the bandwidth is high and delay is low. We use the model to compare the connection-setup costs of these protocols, bounding the possible performance improvement. We evaluate these costs for a range of network characteristics, finding that setup optimizations are relatively unimportant for current modem, ISDN, and LAN users but can provide moderate to substantial performance improvement over high-speed WANs. We also use the model to predict performance over future network characteristics.

## 1 Introduction

The World Wide Web [1] has rapidly become one of the most popular Internet services [2]. The popularity of the web has resulted in a corresponding popularity for HTTP, the standard Hyper-Text Transport Protocol [3, 4]. HTTP is layered over TCP.

The strengths of TCP are well known. TCP is a well understood protocol with carefully tuned flow control and congestion avoidance algorithms [5]. These characteristics make TCP an excellent protocol for bulk data transport in congested networks.

Web traffic is not ideally matched to TCP, however. In practice, web access is request–response oriented with bursts of numerous requests and small, unidirectional responses. Retrieval of a complete web page requires separate requests for text and each embedded image, thus making traffic inherently bursty. Responses are small to keep transmission times down; several studies have documented typical response size as less than 1KB [6], 6KB [7], or 21KB [8]. Finally, web users often bounce rapidly from site to site, as verified by both client [6] and server side traces [8].[1]

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through FBI contracts #J-FBI-95-185 entitled "Cities Online", and #J-FBI-95-204, "Global Operating Systems Technologies". The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Army, DARPA, or the U.S. Government.

The authors can be contacted at 4676 Admiralty Way, Marina del Rey, CA, 90292-6695, or by electronic mail to johnh, katia or touch, each @isi.edu.



[1]For example, from the 1995-96 Boston University survey [6] we can deduce that an upper bound on the mean number of unique URLs read from each site is 15.5. The NCSA server-side traces suggest that clients read a mean of 2.92 text pages at their site per-browsing session.

Unfortunately, TCP is poorly suited to frequent, short, request–response-style traffic. Frequent connection setup and tear-down costs burden servers with many connections left in TIME_WAIT state [9, 10]. Short connections can interact poorly with TCP's slow-start algorithm for congestion avoidance [10]. Finally, TCP's initial three-way handshake adds latency to each transaction [10].

These mismatches between the needs of HTTP and the services provided by TCP contribute to increased latency for most web users. Fundamentally, TCP is optimized for large-scale bulk data transport, while HTTP often needs a light-weight, request–response protocol. Other request–response-style services, including transfer of short e-mail messages or files and RPC protocols, would also benefit from protocol improvements.

Prior work in this field (reviewed in the next section) has identified and proposed solutions to each of these problems. This paper builds upon that work in several ways. First, we present analytic models for HTTP traffic over several transport protocols. These models allow us to compare current and future protocols in a common framework. We can also predict protocol performance on proposed networks. We validate our models by comparing them to measurements from an actual system. Finally, we use the insight provided by the models to reflect on the needs of protocols for HTTP traffic.

This paper makes several contributions to the field. First, we provide a detailed examination of the transient effects of TCP connection start-up. We show that the start-up behavior of TCP's slow-start algorithm depends on the acknowledgment policy of the receiver and that this fact often results in the congestion window opening significantly slower than has often been described. These effects are particularly important for HTTP (and similar short, request–response protocols) where transient effects often dominate performance. Second, we provide an analytic model for web transport over several protocols and use this model to compare these protocols and to validate the experimental results of prior researchers. Third, we apply our model to predict protocol performance in a broad range of network characteristics including future networks characteristics. Finally, we apply these predictions to evaluate the performance of proposed HTTP enhancements. We find that, while proposed enhancements such as persistent HTTP are effective with high-bandwidth network characteristics, they offer much more modest gains in the medium- and low-bandwidth network characteristics common to most Internet users today.

# 2 Related Work

This section summarizes previous and current work in the areas of HTTP performance evaluation as well as transport protocols that have been proposed as alternatives to TCP.

A simplified version of the HTTP over TCP and caching TCP models of this paper can be found elsewhere [11]. That paper focuses on comparison of HTTP with and without persistent connections; this paper more accurately models slow-start and workloads and analyzes additional protocols in more detail.

## 2.1 Persistent-Connection HTTP

Padmanabhan and Mogul conducted experiments to quantify the cost of using TCP as HTTP's transport mechanism [10]. Their examination of a typical HTTP request–response demonstrated throughputs for short responses as small as 10% of the throughput obtainable by bulk data transfers under similar network conditions. They attribute these costs to TCP's connection setup and slow-start mechanisms.

To amortize TCP's connection overhead over multiple HTTP interactions, Padmanabhan and Mogul propose a "persistent-connection" HTTP, or P-HTTP, a variant of HTTP that uses one TCP connection to carry multiple HTTP requests [10]. Mogul also investigates trace-driven simulations of HTTP and P-HTTP, demonstrating that P-HTTP can avoid these setup costs and achieve significantly better performance than HTTP when there is temporal locality in web accesses [12]. By requiring fewer TCP connections than HTTP, P-HTTP also conserves server and network resources.

Padmanabhan and Mogul's results have been corroborated by Simon Spero in an unpublished study [13]. A version of P-HTTP is part of the specification of HTTP/1.1 [4].

Both Padmanabhan and Mogul's and Spero's analyses of HTTP overhead were founded on measurements between relatively well-connected Internet hosts (bandwidth about 1Mb/s, round-trip time 70ms). We show that our analytic model of performance allows us to extend these results to other networks. We validate their results for well-connected hosts; in such cases P-HTTP will improve performance. We also show that when either bandwidth or delay degrade (perhaps due to wide-area congestion, bottleneck links such as a modem or ISDN, or co-location of hosts), then P-HTTP performance improvements are much more modest.

A recent technical note has suggested that use of pipelining is important to get good performance from the HTTP/1.1 implementation of P-HTTP [14]. Pipelining reduces the number of packets transmitted and supports request independence (as discussed in Section 4.1). We discuss the performance implications of HTTP/1.1 with pipelining in Section 5.5.

## 2.2 Transaction TCP

Transaction TCP, or T/TCP [9, 15], was proposed to bridge the gap between the services provided by UDP and TCP for request–response applications.[2] T/TCP improves TCP performance by caching per-host information sufficient to bypass the TCP's three-way handshake and avoid slow start.[3] T/TCP also shortens TCP's TIME_WAIT period from 240 to 12 seconds, reducing the duration that per-connection state is retained.

Stevens compares the time to complete a client-server transaction using TCP, UDP, and T/TCP for different sizes of the request and reply over Pentium-based hardware on a 10Mb/s Ethernet [16]. As expected, the UDP-based client-server yields the smallest latency (11 to 81ms, depending on packet size), and the TCP-based interaction takes the longest to complete (36 to 105ms). In Stevens' experiments, T/TCP is about 5ms more expensive than UDP for a given packet size (and therefore 55–18% faster than TCP).

We extend Steven's results by modeling HTTP traffic over T/TCP. We also show that, with respect to connection establishment costs, HTTP traffic over T/TCP and persistent-connection TCP (P-HTTP over TCP) behave identically.

## 2.3 UDP-Based Request–Response Protocols

The Asynchronous Reliable Delivery Protocol (ARDP) is one example of a reliable message passing protocol built atop UDP for request–response-style interactions between a client and a server. ARDP was proposed and implemented as the transport mechanism for the Prospero information discovery tool [17].

ARDP's main design goal is to provide a reliable yet light-weight communication mechanism to transport requests and responses between clients and servers. The current version of ARDP (in development) borrows TCP-style flow-control, congestion-avoidance, and retransmission algorithms.[4] ARDP avoids TCP's three-way handshake, instead randomly selecting connection identifiers. This approach trades connection setup overhead for a chance of accidental (or intentional) connection identifier reuse.

We will show that although avoiding the three-way handshake is helpful, caching congestion-control information is important for optimal performance.

# 3 Network and Traffic Model

To understand the performance of HTTP over different transport protocols we must characterize the network and the traffic we expect to send. We consider each of these in turn.

[2] A T/TCP "transaction" is a request–response exchange, not a database-style transaction.

[3] While the functional specifications for T/TCP suggest that congestion window be cached, the reference implementation (at ftp://ftp.isi.edu/pub/braden/TTCP.tar.Z) does not cache this value.

[4] Older versions of ARDP have a fixed window size of 16 1250-byte packets and do not do slow-start.

| | |
|---|---|
| *rtt* | round-trip time |
| *bw* | bandwidth |
| *mss* | maximum segment size |
| *stt* | segment-transmission time |
| *muws* | maximum useful window size |

**Table 1: Network characteristics affecting HTTP performance.**

## 3.1 Network Model

Figure 1 shows the beginning of a typical TCP packet exchange. Several parameters are needed to characterize this exchange; we list these in Table 1.

The first three parameters listed in Table 1, round-trip time, bandwidth, and maximum segment size[5] are all properties of a given network path (although observed round-trip time and bandwidth may change due to variations in network and server load). The remaining two parameters can be derived from the others. Segment-transmission time, the time it takes to send the largest possible packet, is directly related to bandwidth and segment size:

$$stt = mss/bw$$

Maximum useful window size is the bandwidth–delay product expressed in an integral number of packets. This final parameter represents the number of segments which must be in flight to keep the network "pipe" full. When the current window size is less than *muws* there will be a delay while acknowledgements return to the sender; when window size is at least *muws* segments then there will be continuous flow of data. Analytically, *muws* is:

$$muws = \lceil rtt/stt \rceil$$

Note that *muws* is the bandwidth–delay product expressed in an integral number of packets.

A final network characteristic not considered here is transmission error rate. We discuss the effects of packet loss on our results when we validate our model in Section 5.3. The primary goal of this paper is to examine startup effects of transport protocols. A complete discussion of the effects of error on transport performance are beyond the scope of this paper, so for the remainder of this paper we assume error-free transmission

[5]Our segment sizes already account for link-level headers.

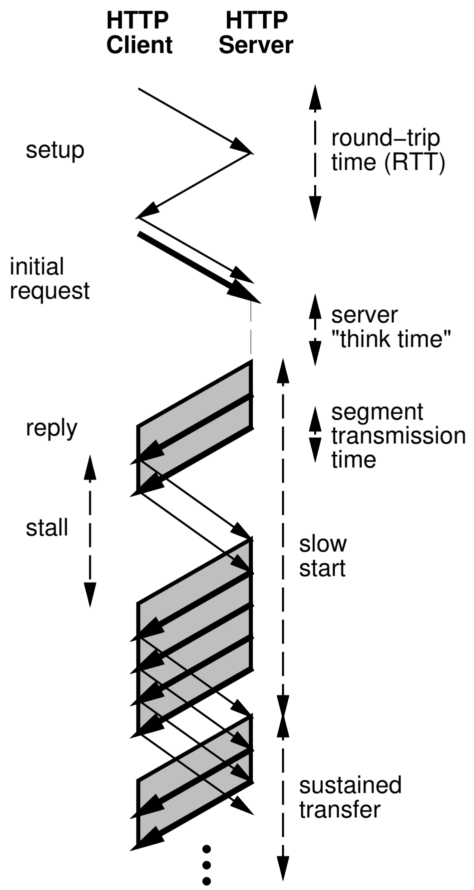


**Figure 1: Packets exchanged in an HTTP over TCP connection not limited by bandwidth.** Bold arrows indicate data transfer while thin arrows show SYN- or ACK-only packets.

Having defined these parameters, we can now quantify them for the networks in which we are interested. These values are given in Table 2.

Our models will use these parameters to predict performance across a range of networks. Where possible we have directly measured these parameters on actual systems. For Ethernet, we use ob-

| **network** | *rtt* | *bw* | *mss* | *stt* | *muws* |
|---|---|---|---|---|---|
| **Ethernet** | 0.7ms | 8.72Mb/s | 1460 B | 1.28ms | 1 pkts |
| **Fast-Ethernet** | 0.7 | 100 | 1460 | 0.111 | 7 |
| **Slow-Internet** | 161 | 0.102 | 512 | 38.5 | 5 |
| **Fast-Internet** | 89 | 1.02 | 512 | 3.85 | 24 |
| **Modem** | 250 | 0.0275 | 512 | 142 | 2 |
| **ISDN** | 30 | 0.122 | 512 | 32 | 1 |
| **WAN-Modem** | 350 | 0.0275 | 512 | 142 | 3 |
| **WAN-ISDN** | 130 | 0.122 | 512 | 32 | 5 |
| **ADSL** | 30 | 6 | 512 | 0.651 | 47 |
| **DirecPC** | 500 | 1 | 512 | 3.91 | 128 |
| **N-Ethernet** | 0.7 | 8.72 | 1460 | 1.28 | 1 |
| **N-Fast-Internet** | 80 | 1.17 | 1460 | 9.52 | 9 |
| **N-Modem** | 150 | 0.0275 | 1460 | 396 | 1 |

**Table 2: Network characteristics for several existing networks. N-Ethernet, N-Fast-Internet, and N-Modem are discussed in Section 5.5.**

served bandwidth and latency measured between two Sun SPARC 20/71 hosts connected by a dedicated 10Mb/s Ethernet. For the Internet we employ two different values, "fast" and "slow" Internet, which corresponds to measured communications speeds between well-connected hosts on the same and different continents, respectively.[6] Actual Internet performance represents a continuum between these points. The fast-Internet case corresponds roughly to characteristics present in Mogul's and Spero's studies [12, 13]. DirecPC also presents measured values from a system with satellite down-link and a modem back-channel [18]. (We assume that the back-channel is not a factor limiting performance.)

For several other networks we have had to estimate these parameters. Modem and ISDN figures employ measured latencies and theoretical bandwidths. Fast-Ethernet and ADSL use theoretical bandwidths and latencies suggested by similar systems (10Mb/s Ethernet and ISDN, respectively). We look forward to replacing these estimates with actual measurements as these systems become available.

Finally, only the Slow- and Fast-Internet figures consider wide-area limitations on latency and bandwidth; other cases assume that the client is directly connected to the server by the given networking technology. We can reflect wide-area limitations by adding latency and capping bandwidth to that observed in the Slow- and Fast-Internet cases. For Modem and ISDN we therefore show WAN version with 100ms additional latency. For faster technologies (ADSL, DirecPC, Slow- and Fast-Ethernets) bandwidth and latency can be approximated by the Slow- to Fast-Internet cases.

We can already observe that *muws* is fairly low in many current networks, with the exception of the fast-Internet case. Once the transmission window has opened up past this value, acknowledgments of outstanding packets are returned as fast as packets are transmitted. *muws* is directly related to protocol overhead; we will show later that when it is small, connection setup optimizations have little to optimize and so provide performance similar to HTTP over TCP.

## 3.2 Traffic Model

Performance of transport protocols also depends upon the traffic characteristics of HTTP. We consider several potential HTTP workloads:

**small page** A single 5KB web page.

**medium page** A single 25KB web page.

[6] For the fast-Internet case we measured communications between darkstar.isi.edu and prep.ai.mit.edu, for the slow case, between darkstar.isi.edu and ftp.connect.org.uk. These measurements were taken on a Saturday afternoon (7 hours west of UTC), 11 May 1996.

**large page** A single 100KB web page.

**small cluster** A single 6,651B page with embedded 3,883B and 1,866B images.[7]

**medium cluster** A single 3,220B page with three embedded images, sizes 57,613B, 2,344B, and 14,190B.[8]

**large cluster** A single 100KB page with 10 embedded 25KB images.

Notice that at each change in size (from small to medium to large), the total amount of data exchanged is about five times larger.

Each of the cluster workloads requires multiple HTTP requests, one per page or image. In multi-request exchanges we assume that all requests are independent. Requests A and B are independent if request B can be initiated before request A has completed. Although a simple client program would sequentially request each item, modern multi-threaded browsers initiate multiple parallel image requests as the basic HTML page is received, thus allowing some degree of request independence.

Choice of traffic will influence our protocol evaluation. The small and medium cases are representative of typical web pages today. We consider the large cases representative of what may become common in the future as networks with higher bandwidth become more widely available.

Finally, we also need to model the size of HTTP requests and the time it takes a server to parse and interpret the request (server processing time). To simplify our model we assume a constant request size of 256B and zero processing-time. A more complex model of request size is not warranted since requests almost always fit in one segment and so performance is dominated by response size.[9] The zero processing-time assumption is clearly incorrect; request-handling overhead depends strongly on server hardware, software, and load. We remove this effect from our computations to focus instead on the network protocol aspects of HTTP instead of server implementation. We reconsider processing-time when we validate our model in Section 5.

[7]The front page (`http://www.yahoo.com`) at Yahoo on May 1, 1996.

[8]The front page (`http://www.gnn.com`) at GNN on May 1, 1996.

[9]Some older browsers had substantially longer requests [13]. Performance concerns are one reason modern browsers keep requests short.

# 4 Protocol Analysis

We next examine protocol performance for the networks and traffic patterns we are considering. We examine the interactions between HTTP and several classes of protocols:

**TCP** HTTP currently runs over TCP, opening a new connection for each transaction.

**Connection caching protocols** To avoid connection setup overheads, P-HTTP and T/TCP cache connections or connection information across multiple requests.

**UDP-based request–response protocols** ARDP employs UDP to avoid TCP setup costs.

To examine these protocols against a consistent baseline we first consider a lower bound on transaction time.

## 4.1 Minimum Transmit Times

Logically, the minimum possible transaction time is the one round-trip time delay inherent in communication, plus the time it takes to send the request and receive the reply, and any time spent at the server:

$$\begin{aligned} T_{min} &= rtt \\ &\quad + req_{min} \\ &\quad + processing \\ &\quad + reply_{min} \\ req_{min} &= req_{size}/bw \\ reply_{min} &= reply_{size}/bw \end{aligned} \tag{1}$$

This equation is straightforward and can be reduced to data size divided by bandwidth. We present it in this detail to illustrate the differences and similarities between different transport protocols.

A series of $n$ independent requests will incur only one round-trip latency because they are pipelined. The total required time will therefore be:

$$S_{min} = rtt + \sum_{i=1}^{n} (T_{min}(i) - rtt) \quad (2)$$

The assumption of independent requests implies a browser which is multi-threaded or which pipelines requests and that the subsequent requests are sent immediately without waiting for the prior response. (In particular, we assume that the second request can be made before the complete results of the first request have returned.) If we were to assume a single-threaded browser making $n$ sequential requests we would then add an additional $(n-1) \times rtt$ delay to Equation 2 while the client determines and requests the next page. If we assume that no additional requests could be made until the first was completed (as would be the case in a multi-threaded browser where all image references are in the last segment of the first page), we would add one additional *rtt*.

Because of our assumptions about request size and processing-time, the primary factor influencing minimal transmission times will be $reply_{min}$. Table 3 summarizes the minimum possible transmission times for the networks and workloads we consider.

## 4.2 Simple Model

We can construct a very simple estimate of when transport protocol overhead will be significant by comparing the ratio of the bandwidth–delay product to the transaction size. In any streaming transport protocol several round-trip exchanges are required to reach steady-state performance (assuming networking conditions are not known *a priori*). When the offered transaction is too small, stability is not achieved, and transient effects are amplified.

We can approximate the minimum amount of time a connection would have to stabilize by comparing the ratio of transmitted data to pipe size. For an HTTP transaction, assuming that $req_{size}$ is zero, this ratio is:

$$\frac{reply_{size}}{bw \times rtt} \quad (3)$$

When this ratio is small we would expect protocol setup costs to dominate performance; when it is large setup costs would be amortized.

An alternate view of the same concept inverts this ratio to get the pipe size in reply-sized units.

$$\frac{rtt}{reply_{size}/bw} \quad (4)$$

This equation is a good approximation for one round-trip overhead per reply (the exact value would be $rtt/T_{min}$). We can use this equation to provide a first approximation for setup overheads. To estimate overhead for single page retrievals we apply this equation directly. For clusters of retrievals we use the harmonic mean[10] of ratios for each retrieval if each retrieval requires a separate connection. If connection overhead is amortized across all of a cluster's replies (as it would be if retrievals were batched over a single connection) we treat all replies as a single large response.

Table 4 shows these ratios, highlighting exchanges where the overhead approximation exceeds 25%. These simple equations provide a good predictor of where transient effects will be high, but they fail to accurately capture these effects when the bandwidth/delay product rises. To accurately estimate performance of actual network protocols in these cases we next consider the effects of congestion avoidance protocols.

## 4.3 HTTP over TCP

We next consider the overhead present in TCP when compared to the minimum transaction time. TCP adds several sources of overhead: protocol headers, the three-way handshake at connection setup, the slow-start algorithm, and retransmissions and congestion control delay due to packet loss. Packet loss rates depend on a number of factors beyond the scope

[10] The harmonic mean of $n$ values is $n/(\sum 1/x_i)$.

| network | small-page | medium-page | large-page | small-cluster | medium-cluster | large-cluster |
|---|---|---|---|---|---|---|
| **Ethernet** | 5.4ms | 23.3ms | 90.5ms | 12.2ms | 69.3ms | 317ms |
| **Fast-Ethernet** | 1.11 | 2.67 | 8.53 | 1.7 | 6.68 | 28.3 |
| **Slow-Internet** | 565 | 2100 | 7870 | 1150 | 6050 | 27300 |
| **Fast-Internet** | 129 | 283 | 860 | 188 | 678 | 2800 |
| **Modem** | 1740 | 7430 | 28800 | 3910 | 22000 | 101000 |
| **ISDN** | 366 | 1650 | 6450 | 853 | 4930 | 22600 |
| **WAN-Modem** | 1840 | 7530 | 28900 | 4010 | 22100 | 101000 |
| **WAN-ISDN** | 466 | 1750 | 6550 | 953 | 5030 | 22700 |
| **ADSL** | 36.8 | 62.9 | 161 | 46.7 | 130 | 489 |
| **DirecPC** | 541 | 697 | 1280 | 600 | 1100 | 3260 |

**Table 3: Minimal theoretical times to send different workloads across different networks.**

| network | small-page | medium-page | large-page | small-cluster | medium-cluster | large-cluster |
|---|---|---|---|---|---|---|
| **Ethernet** | 0.16 | 0.03 | 0.01 | 0.19 / 0.06 | 0.04 / 0.01 | 0.02 / 0.00 |
| **Fast-Ethernet** | *1.79** | *0.36** | 0.09 | *2.22*/ 0.74** | *0.47*/* 0.12 | *0.28*/* 0.03 |
| **Slow-Internet** | *0.42** | 0.08 | 0.02 | *0.52*/* 0.17 | 0.11 / 0.03 | 0.07 / 0.01 |
| **Fast-Internet** | *2.31** | *0.46** | 0.12 | *2.87*/ 0.96** | *0.61*/* 0.15 | *0.36*/* 0.03 |
| **Modem** | 0.18 | 0.04 | 0.01 | 0.22 / 0.07 | 0.05 / 0.01 | 0.03 / 0.00 |
| **ISDN** | 0.09 | 0.02 | 0.00 | 0.12 / 0.04 | 0.02 / 0.01 | 0.01 / 0.00 |
| **WAN-Modem** | 0.25 | 0.05 | 0.01 | *0.30*/* 0.10 | 0.07 / 0.02 | 0.04 / 0.00 |
| **WAN-ISDN** | *0.41** | 0.08 | 0.02 | *0.50*/* 0.17 | 0.11 / 0.03 | 0.06 / 0.01 |
| **ADSL** | *4.61** | *0.92** | 0.23 | *5.71*/ 1.90** | *1.22*/ 0.30** | *0.72*/* 0.07 |
| **DirecPC** | *12.80** | *2.56** | *0.64** | *15.86*/ 5.29** | *3.39*/ 0.85** | *2.01*/* 0.18 |

**Table 4: Approximation of one round-trip overhead per transaction** (Equation 4). Highlighted values indicate ratios more than 0.25 where transient effects may dominate performance. For cluster workloads ratios are given assuming separate and single connections.

of this paper; we therefore consider only the first three sources of overhead. We discuss how packet loss would impact our model in Section 5.3.

An idealized packet trace for a request–response transaction over TCP is shown in Figure 1. In this packet trace we can see how the three-way handshake (labeled setup) adds one round-trip time overhead. We next consider the effects of TCP's slow-start algorithm.

### 4.3.1 TCP slow-start

TCP's slow-start algorithm limits transmission by a congestion window (*cwnd*) which is initialized to one segment and increases each time an ACK is received [5]. The size of increase changes: initially *cwnd* grows in one segment increments (the slow-start phase), then later by $1/cwnd$ (congestion avoidance phase). TCP is thus "clocked" by the flow of acknowledgments.

For high bandwidth/delay-product paths, TCP initially alternates between segment transmission and stalls waiting for ACKs from these segments to return. The number of segments sent per stall increases exponentially during slow-start until enough segments are in flight that ACKs return continuously. To model slow-start behavior we therefore need to know how many segments are sent between stalls, how much time each stall wastes, and how many stalls occur until steady state is reached or transmission ends.

We originally expected that the number of packets between each stall would follow a simple exponential pattern: 1, 2, 4, 8, and so on. Modern TCP implementations deviate from this behavior for two reasons. First, in BSD-derived TCP implementations the ACK of the SYN packet on the HTTP server opens the congestion window, so the *cwnd* for the reply begins at 2. Second, TCP's delayed-acknowledgment algorithm normally causes the client to ACK every other segment, not each segment [19]. Because the congestion window opens per ACK received rather than per segment acknowledged, the slow-start window opens much slower than is usually assumed. We develop the exact relationship and review the details of slow-start and delayed acknowledgments in Appendix A. Table 5 summarizes our findings for three different acknowledgment policies. The rightmost column illustrates our original expectations, the left column shows a lower-bound on slow-start performance.

| stall | no delayed ACKs | delayed ACKs | ACK every segment |
|---|---|---|---|
| 1 | 2 (2) | 2 (2) | 2 (2) |
| 2 | 3 (5) | 3 (5) | 4 (6) |
| 3 | 3 (8) | 5 (10) | 8 (14) |
| 4 | 6 (14) | 8 (18) | 16 (30) |
| 5 | 9 (23) | 12 (30) | 32 (62) |
| 6 | 12 (35) | 18 (48) | 64 (126) |
| 7 | 18 (53) | 27 (75) | 128 (254) |
| 8 | 27 (80) | 41 (116) | 256 (510) |
| 9 | 42 (122) | 62 (178) | 512 (1022) |
| 10 | 63 (185) | 93 (271) | 1024 (2046) |

**Table 5: Number of segments between slow-start stalls for different acknowledgment policies.** The cumulative number of segments sent is given in parentheses. These columns represent $segs_{nda}(i)$ ($csegs_{nda}(i)$), $segs_{da}(i)$ ($csegs_{da}(i)$), and $segs_{ae}(i)$ ($csegs_{ae}(i)$) from Appendix A, where $i$ is the stall number.

Not all the time of each stall is completely wasted: an increasing part of each stall is spent sending packets, until *cwnd* opens past *muws*. Appendix A quantifies this cost with the formula $slowstart_{TCP}$. In the next section we use this result to develop HTTP over TCP transaction time.

### 4.3.2 Performance and discussion

We can summarize the cost of accessing an object via HTTP over TCP by including the extra round-trip of the setup and the slow-start costs:

$$\begin{aligned} T_{TCP} &= 2 * rtt \\ &\quad + req_{min} \\ &\quad + processing \\ &\quad + reply_{TCP} \\ reply_{TCP} &= slowstart_{TCP} \\ &\quad + reply_{min} \end{aligned} \tag{5}$$

| network | model | small-page | medium-page | large-page | small-cluster | medium-cluster | large-cluster |
|---|---|---|---|---|---|---|---|
| **Ethernet** | **TCP** | 1.13 | 1.03 | 1.01 | 1.17 | 1.04 | 1.02 |
| | **caching** | 1.13 | 1.03 | 1.01 | 1.06 | 1.01 | 1.00 |
| | **ARDP** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **Fast-Ethernet** | **TCP** | *2.16** | *1.62** | 1.19 | *2.92** | *1.79** | *1.64** |
| | **caching** | *2.16** | *1.62** | 1.19 | *1.76** | 1.19 | 1.06 |
| | **ARDP** | *1.53** | *1.36** | 1.11 | *1.69** | *1.37** | *1.37** |
| **Slow-Internet** | **TCP** | *1.50** | 1.13 | 1.04 | *1.74** | 1.19 | 1.11 |
| | **caching** | *1.50** | 1.13 | 1.04 | 1.25 | 1.05 | 1.01 |
| | **ARDP** | 1.22 | 1.06 | 1.02 | *1.32** | 1.08 | 1.05 |
| **Fast-Internet** | **TCP** | *2.94** | *2.11** | *1.36** | *4.60** | *2.55** | *2.23** |
| | **caching** | *2.94** | *2.11** | *1.36** | *2.34** | *1.37** | 1.11 |
| | **ARDP** | *2.26** | *1.79** | *1.26** | *3.18** | *2.03** | *1.88** |
| **Modem** | **TCP** | 1.14 | 1.03 | 1.01 | 1.19 | 1.05 | 1.03 |
| | **caching** | 1.14 | 1.03 | 1.01 | 1.06 | 1.01 | 1.00 |
| | **ARDP** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **ISDN** | **TCP** | 1.08 | 1.02 | 1.00 | 1.11 | 1.02 | 1.01 |
| | **caching** | 1.08 | 1.02 | 1.00 | 1.04 | 1.01 | 1.00 |
| | **ARDP** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **WAN-Modem** | **TCP** | *1.30** | 1.07 | 1.02 | *1.42** | 1.10 | 1.06 |
| | **caching** | *1.30** | 1.07 | 1.02 | 1.14 | 1.03 | 1.01 |
| | **ARDP** | 1.11 | 1.03 | 1.01 | 1.16 | 1.04 | 1.02 |
| **WAN-ISDN** | **TCP** | *1.49** | 1.13 | 1.03 | *1.72** | 1.18 | 1.11 |
| | **caching** | *1.49** | 1.13 | 1.03 | 1.24 | 1.05 | 1.01 |
| | **ARDP** | 1.21 | 1.06 | 1.01 | *1.31** | 1.08 | 1.05 |
| **ADSL** | **TCP** | *3.37** | *3.12** | *1.83** | *6.01** | *4.03** | *3.99** |
| | **caching** | *3.37** | *3.12** | *1.83** | *2.87** | *1.67** | *1.27** |
| | **ARDP** | *2.56** | *2.64** | *1.64** | *4.08** | *3.10** | *3.32** |
| **DirecPC** | **TCP** | *3.74** | *4.44** | *3.36** | *7.60** | *7.57** | *9.30** |
| | **caching** | *3.74** | *4.44** | *3.36** | *3.47** | *2.35** | *1.93** |
| | **ARDP** | *2.82** | *3.72** | *2.97** | *5.10** | *5.75** | *7.61** |

**Table 6: Ratios of predicted protocol transaction times to minimum transaction time** for different protocol models, workloads, and network characteristics. Ratios are for the HTTP over TCP model ($S_{TCP}/S_{min}$); HTTP over TCP with connection caching, assuming no cache hit ($S_{cache\text{-}miss}/S_{min}$, with cache hits the performance ratio is always 1); and HTTP over ARDP ($S_{ARDP}/S_{min}$). Highlighted values indicate overheads larger than 0.25.

The cost of a series of independent requests is then:

$$S_{TCP} \quad = \quad rtt + \sum_{i=1}^{n} (T_{TCP}(i) - rtt) \qquad (6)$$

We can now define the overhead of HTTP over TCP as:

$$overhead_{TCP} \quad = \quad S_{TCP}/S_{min} \qquad (7)$$

The TCP lines of Table 6 show $overhead_{TCP}$ (given by Equation 7) for a variety of workloads and network characteristics (we describe the other lines of this table in the following sections). To highlight significant overhead we indicate ratios higher than 1.25 with italics.

We can draw several conclusions from these results. First, overhead for the Ethernet, modem, and ISDN networks is reasonable (less than 25% overhead) for all workloads (although adding WAN delays raises small-page and small-cluster overheads to significant levels). These networks have a *muws* at most 2 and so do not pay any slow-start penalty. The extra round-trip cost of the three-way handshake is proportionally largest for short responses, but even there, overhead is less than 25%.

On the other hand, network such as Fast-Ethernet, Fast-Internet, ADSL, and DirecPC have substantially higher overheads because they have much higher *muws* sizes. The high bandwidth–delay product of these networks requires 7–128 segments in-transit to fill the communications "pipe"; for small files, substantial time is spent waiting for packet acknowledgments. These overheads are most pronounced when smaller workloads are processed, since larger workloads are able to expand the congestion window and amortize the connection setup cost over the duration of the connection.

Local-area networks behave differently than wide-area networks in two ways that affect performance. First, many TCP implementations disable slow-start on LANs where congestion control is done at the link layer. Second, most LANs allow TCP segments much larger than are typical in WANs (1460 bytes instead of 512 or 536 bytes), although wide deployment of path-MTU discovery supports larger WAN segment size [20]. An interesting observation is that employing slow-start does not substantially affect performance over 10Mb/s Ethernet because large segment-size and low round-trip time result in a small *muws*.

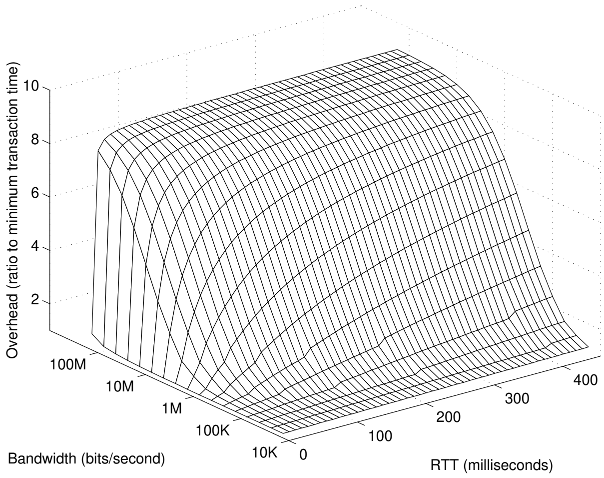


**Figure 2: TCP congestion avoidance overhead for various bandwidth–delay products ($S_{TCP}/S_{min}$), for a 512B segment size and the small-cluster workload.**

To explore the effects of these overheads for a wider variety of network characteristics, Figure 2 examines a single workload (the small-cluster workload) with a fixed segment size (512B) and a range of bandwidths and round-trip times.

Again, we observe in this graph that overhead is fairly low when the bandwidth–delay product (and so *muws*) is small, either because of small bandwidths or small round-trip-times. Overhead rises sharply when the product of these values grows. Finally, overhead tops out at about 9 times the minimal transfer time.

This workload-dependent limit indicates network characteristics where all transmission time for all transactions is spent in the slow-start realm. (For the medium-cluster case the maximum possible overhead is about 16 times minimal, for large-cluster it is about 55 times minimal.)

Note that these high overheads occur only when the bandwidth–delay product is very large, as in a satellite connection. It is well known that TCP is not the best protocol for such situations. When either bandwidth is low or delay is very low, TCP performs much better.

### 4.4 HTTP over TCP with connection caching

P-HTTP [10] has been proposed as a solution to several problems resulting from running HTTP over TCP, and persistent connections are part of the HTTP/1.1 standard [4]. P-HTTP reuses a single TCP connection to amortize connection setup and congestion avoidance costs across several HTTP requests. When a transaction is completed, the TCP connection is left open for subsequent transactions. This connection is reused for subsequent requests; it will be closed if server or client demand is high, or when idle for a given length of time.

Stevens has suggested the use of T/TCP for HTTP traffic [16]. T/TCP [9] is an extension of TCP enhanced to support transactions. T/TCP caches TCP connection setup information so that subsequent TCP connections avoid the three-way handshake and reduce slow-start cost. Thus, like P-HTTP, T/TCP pays the full cost of TCP setup for the first connection and avoids setup costs for subsequent connections. Cached T/TCP information will be flushed using algorithms similar to those for breaking P-HTTP connections (for example, using a least-recently used or a periodic time-out).

Although P-HTTP provides connection caching at the user level and T/TCP does so in the kernel, a series of requests in either protocol performs identically (at the level of detail we consider here). In each protocol the first request pays the full cost to open a standard TCP connection, but subsequent requests avoid the three-way-handshake and continue to develop the slow-start window. We therefore model these protocols together as "HTTP over caching TCP" protocols.

There are two possible costs for caching protocols, with and without cache hits:

$$T_{\textit{cache-miss}} = T_{TCP} \tag{8}$$

$$T_{\textit{cache-hit}} = T_{min} \tag{9}$$

In a series of requests to a new server the first will always be a miss and subsequent requests will usually be hits.

To quantify a series of requests we make the simplifying assumption that after the first exchange the congestion window will be completely opened. For the purpose of our model, the congestion window is fully opened when $cwnd \geq muws$, since opening the window further will not impact performance. Therefore, for our assumption to be true, $cwnd(stalls(\lceil reply_{size}/mss\rceil)) \geq muws$ after the first exchange, where $reply_{size}$ is the size of the first reply from the server. (We analyze *cwnd* and *stalls* in Appendix A.) Our workloads satisfy this assumption for the networks listed in Table 2 satisfy this assumption.

Given this assumption, all requests after the first will be cache hits under conditions of normal load (under high load the client or server may terminate connections causing additional cache misses). The first request may or may not be a cache hit depending on the user's browsing pattern and server load. We therefore define two formulas for series of independent accesses, one assuming that the first request is a cache hit and one assuming that it's not:

$$\begin{aligned} S_{\textit{first-miss}} &= T_{\textit{cache-miss}}(1) \\ &\quad + \sum_{i=2}^{n}(T_{\textit{cache-hit}}(i) - rtt) \end{aligned} \tag{10}$$

$$\begin{aligned} S_{\textit{first-hit}} &= rtt + \sum_{i=1}^{n}(T_{\textit{cache-hit}}(i) - rtt) \\ &= S_{min} \end{aligned} \tag{11}$$

Finally, overhead is again:

$$overhead_{\textit{first-miss}} = S_{\textit{first-miss}}/S_{min} \tag{12}$$

$$\begin{aligned} overhead_{\textit{first-hit}} &= S_{\textit{first-hit}}/S_{min} \\ &= 1 \end{aligned} \quad (13)$$

The caching lines of Table 6 show the performance results of our workloads assuming that the first web page access does not use a cached connection ($overhead_{\textit{first-miss}}$ in Equation 12). Note that for the cluster workloads, accesses after the first are cache hits.

Several observations about HTTP over connection-caching TCP protocols are apparent from this table. First, HTTP over caching TCP performance is the same as standard HTTP over TCP performance for single page queries. Second, caching-TCP performance is somewhat better than standard TCP for the cluster cases because connections after the first need not pay for three-way handshake or slow-start. We explore this relationship further in Section 6. Finally, overhead is still high for the Fast-Ethernet and Fast-Internet cases with cluster workloads. In these cases, the large bandwidth–delay product results in significant overhead while the congestion window is opened even when amortized over an entire series of connections.

Finally, if we assume that the first transaction results in a cache-hit for a caching-TCP protocol (Equation 13), then caching-TCP has no overhead. Thus, when caches last long enough to encompass access of multiple clusters, caching protocols are very helpful. Implementation issues can limit this benefit, however [21].

## 4.5 HTTP over Multiple, Concurrent TCP Connections

Many web browsers open multiple concurrent connections to mitigate TCP start-up costs (HTTP over parallel connections). We can bound their performance by HTTP over TCP with and without connection caching. Our HTTP-over-TCP model overestimates transmission time by not considering parallelism in the concurrent slow-start of each connection. HTTP over connection caching underestimates transmission time by assuming that there is no penalty for slow-starts of later requests in a cluster. A better approximation might be obtained by treating *rtt* as if it were $rtt/n$, for $n$ concurrent connections. Completely specifying behavior with multiple parallel connections is an area of continuing work.

## 4.6 HTTP over UDP-Based Protocols

Since web access is typically request–response oriented, we examine the performance of HTTP over a request–response style protocol such as ARDP. ARDP avoids TCP's three-way handshake, while it keeps TCP's slow-start algorithm for congestion avoidance. Therefore, the time to complete a HTTP transaction over ARDP is:

$$\begin{aligned} T_{ARDP} &= rtt \\ &\quad + req_{ARDP} \\ &\quad + processing \\ &\quad + reply_{ARDP} \\ req_{ARDP} &= req_{min} \\ reply_{ARDP} &= slowstart_{TCP} \\ &\quad + reply_{size}/bw \end{aligned} \quad (14)$$

The total time to complete a series of independent HTTP requests is given by:

$$S_{ARDP} = rtt + \sum_{i=1}^{n} (T_{ARDP}(i) - rtt) \quad (15)$$

and the overhead by

$$overhead_{ARDP} = S_{ARDP}/S_{min} \quad (16)$$

The ARDP lines of Table 6 show $overhead_{ARDP}$ (Equation 16 for the different workloads and network characteristics). Note that for the Ethernet, modem, and ISDN networks, HTTP transactions over ARDP result in minimal transaction times. This confirms that because of their small maximum useful window size (*muws*), these networks do not pay any slow-start penalty.

On the other hand, ARDP's overhead becomes noticeable in the higher bandwidth–delay-product cases (Fast-Ethernet, both Internets, ADSL and DirecPC). ARDP also incurs higher overhead than TCP with connection caching for the cluster workloads. This overhead is due to the fact that ARDP always slow-starts, while caching the connection setup parameters allows the caching protocols to avoid slow-start every time.

Avoiding the three-way handshake is especially helpful for single, brief request–response interactions. For a series of requests to the same server, though, ARDP performance suffers because we do not cache congestion information between calls. As a result of this observation we plan to provide simple congestion-information caching in a future version of ARDP.

# 5 Validation

To relate the analytic results of the prior section to real-world performance we next validate them using traces of actual HTTP traffic. This validation has three goals: first, to show that we model aspects of the protocol relevant to performance. Second, to show that comparisons between the modeled protocols are valid. Finally, we will look for areas where implementations can be improved.

Since it would be impractical to validate each of the 150 combinations of workload, network, and the protocols described in this paper, we instead consider only four cases: HTTP over simple and caching TCP transport protocols with the small-cluster workload and Ethernet and Fast-Internet networks.

## 5.1 Methodology

Our experiments consisted of the four combinations of Ethernet and Fast-Internet networks and HTTP over simple and caching TCP protocols. In all cases our server computer was a Sun SPARC model 20/71 running SunOS 4.1.3 with some TCP/IP modifications (IP-multicast support and a 16KB default TCP window size). We describe server software and client hardware configurations below. In all cases our HTTP client was a custom-written Perl script retrieving the small-cluster workload.[11] We also logged all relevant TCP traffic on the server's network.

For experiments over Ethernet, the client computer was a Sun4-20/71 identical to the server. These computers were connected by a dedicated 10Mb/s Ethernet. Note that SunOS bypasses TCP slow-start when both computers are on the same physical network. We wanted to measure the effects of standard TCP in a high-bandwidth, low-latency environment rather than that of a particular TCP implementation, so we removed this optimization for our experiments.

For the Fast-Internet experiments the client computer was a Sun SPARC-20 running unmodified SunOS 4.1.3. Measurements between the server (located in Los Angeles) and the client (in Washington, D.C.) were taken over the Internet with evening (U.S. West-coast time) background traffic present. Average round-trip time was 133ms and bandwidth was 0.734Mb/s (as measured by repeated FTP of a 1.5MB file) over the 11 hops between the client and our server at ISI.

Our implementation of HTTP over simple TCP was HTTP/1.0 with an Apache 1.0.5 server. The client made HTTP/1.0-style requests.

For HTTP over caching TCP protocols we used the fourth beta version of Apache 1.1 with some modifications. This server implements "keep-alive" HTTP connections, an experimental implementation of persistent connection HTTP (abbreviated HTTP/1.0+KA) similar in character to persistent connections recently standardized as HTTP/1.1 [4]. This server was slightly modified to avoid two interactions between P-HTTP and TCP which substantially reduce performance [21]. Our client made HTTP/1.0-style requests with the "Connection: Keep-Alive" header; the server returned MIME-style headers and page contents with page size determined by the Content-Length header.

[11] Although the program is interpreted, we have verified that it can saturate our Ethernet and so does not pose a performance bottleneck.

## 5.2 Slow-Start Validation

We have observed that a key aspect of HTTP over TCP performance is slow-start behavior. Slow-start performance is dependent upon the client's ACK rate; when a client acknowledges every other segment with delayed acknowledgments the congestion window opens much more slowly than if every segment is explicitly acknowledged. Table 5 summarizes these effects based upon bounds of the slow-start rate developed in Appendix A.

To validate that our bounds on the slow-start rate are accurate, we examined a number of packet traces of HTTP over caching TCP (HTTP/1.0+KA) and FTP traffic between Los Angeles and Washington, D.C., hosts. Figure 3 shows a representative plot of the packets exchanged for the HTTP/1.0+KA requests for the small-cluster workload. As can be seen from this graph, the round-trip time is about 133ms and the client acknowledges every other packet. From the pattern of ACKs we can infer that no timeout-induced delayed acknowledgments occurred in these transactions.

To validate our slow-start model we will examine two parts of the total exchange, the first request (from time 0 to 0.7s) and the second (from 0.7 to 0.9s).

In the first request we see the pattern predicted by the no-delayed-acknowledgment analysis of Table 5: 2, 3, 3, and 6 segments, each with an $\sim 1$ *rtt* delay stall between them. (Note that in the 6-segment stall the sixth segment advances the sequence number by only 19 bytes and so is plotted nearly on top of the fifth segment, and that the ack for this segment was piggybacked on the next request.) From this we conclude that, in the absence of delayed acknowledgments, we correctly predict segments per stall.

In BSD implementations of TCP the delayed ACK timer fires every 200ms, independent of packet arrival. We expect that delayed ACKs will speed up opening of the slow-start window by causing occasional single-segment ACKs instead of delays until two packets have been received. In practice we observe that delayed ACKs sometimes alter both the pattern of packet transmission between stalls and the stall delay. For the small-cluster workload, delayed ACKs seem to have little effect on overall performance.

In the second request we see back-to-back transmission of all nine segments (again, the final segment is short and is obscured by the previous segment on the plot). This behavior is consistent with our model of the congestion window; the window started at two and was opened by one for each of the seven acknowledgments received.

Based on analysis of packet traces from which these examples are drawn we conclude that our lower bound for segments-per-stall is accurate assuming that no delayed ACKs are triggered.

## 5.3 Model Adjustments

Our model focuses on transport-layer issues and therefore makes several simplifying assumptions about client and server software. To validate our results we must account for these assumptions. In particular:

**server processing-time** Our basic model assumes zero processing-time. In our experiments we observe an average 3.7ms server processing-time, so we must add 3.7ms per page to our adjusted model.

**request independence** Our basic model assumes that all requests are independent. Unlike modern browsers, our simple client is not multi-threaded and so issues dependent requests (as described in Section 3.2). We can see two such stalls (marked "dependent delay") in Figure 3. We correct for this artifact by adding 1 *rtt* delay per page after the first.

**inexact bandwidths** Our basic model assumed that Fast-Internet bandwidth was 1Mb/s with 89ms *rtt*. We selected these characteristics to emulate experimental characteristics of other researchers. Our validation experiments instead observed a bandwidth of 0.734 Mb/s and a *rtt* of 133ms; we correct for this using observed network characteristics in our revised estimate.

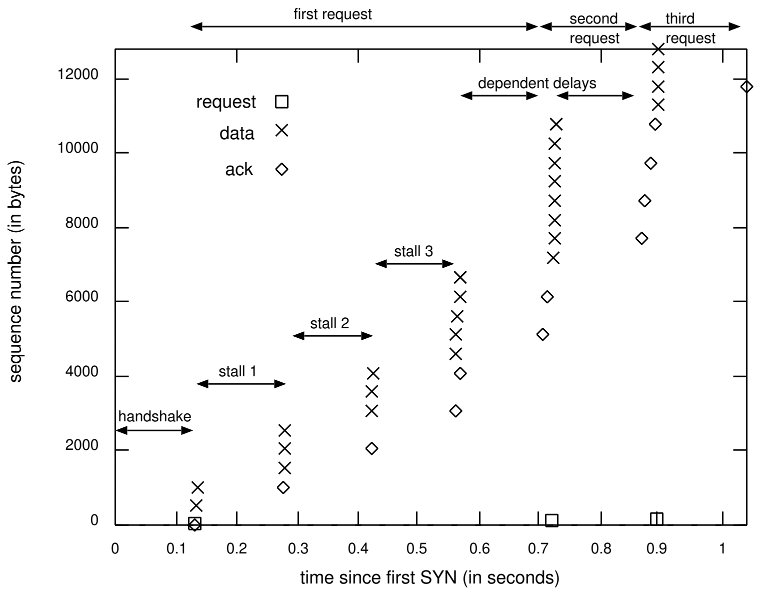


**Figure 3: A packet trace of the HTTP over caching TCP (HTTP/1.0+KA) requests for the small-cluster workload.**

Table 7 shows our original predictions and our predictions adjusted for these factors. We interpret these results in the next section.

Finally, our model assumes no packet loss. We believe that we experienced no packet loss in our Ethernet experiments and loss in only one of the fast-Internet transactions. Packet loss depends heavily on congestion. Because packet loss is detected by timeout and causes re-evaluation of congestion control, even low packet loss rates can cause measured values substantially longer than predicted by our model. In such cases, connection setup costs would likely be overwhelmed by loss-recovery, and therefore, the benefits of connection caching protocols less noticeable.

## 5.4 Model Validation and Discussion

We undertook this validation with two goals: to compare our analysis to reality and to insure that comparisons among our analytic results are valid. We evaluate these goals below. In addition, in the process of validation we found several interactions between P-HTTP and TCP which substantially reduce performance.

Except for the case of HTTP/1.0 over Ethernet, our validation suggests that the model, when adjusted, is accurate within 5% of measured values. The HTTP/1.0 over Ethernet case shows a discrepancy of about 40%. A high bandwidth and low delay link (like Ethernet) makes modeled network overhead small, so other kinds of overheads (which are not modeled) can be noticeable. Furthermore, the modeled overhead is very sensitive to latency at high bandwidths as is shown in Figures 2 and Figure 4.

Another source of error in our model results from interactions between the application-level behavior and the underlying TCP implementation. In the course of validation we found two such interactions that crippled HTTP/1.0+KA performance [21]. In both cases, packets shorter than maximum-segment-size caused our TCP connection to wait for a delayed acknowledgment, stalling data transfer for up to 200ms. We worked around both these problems with small application-level changes, eliminat-

| protocol | implement. | network | prediction basic | adjusted | measured | | ratio m:a |
|---|---|---|---|---|---|---|---|
| TCP | HTTP/1.0 | Ethernet | 12.8ms | 26.8ms | 36.8ms | (10ms, ±2.0ms) | 1.37 |
| caching-TCP | HTTP/1.0+KA | Ethernet | 11.4 | 25.4 | 26.6 | (8.8, ±1.7) | 1.05 |
| TCP | HTTP/1.0 | Fast-Internet | 977 | 1730 | 1716 | (101, ±20.1) | 0.99 |
| caching-TCP | HTTP/1.0+KA | Fast-Internet | 536 | 1070 | 1103 | (48, ±9.5) | 1.03 |

**Table 7: Validation experiments for our models.** All experiments used the small-cluster workload. *Basic* indicates our basic (unadjusted) model, *adjusted* is the model corrected as described in Section 5.3, *measured* indicates the average over 100 trials with the standard deviation and 95% confidence intervals given in parentheses, *ratio m:a* shows the ratio of measured to prediction/adjusted times.

ing these sources of error. Other interactions between the application-level interface and our TCP implementation result in the transmission of short segments. We do not believe that this interaction causes the catastrophic performance loss observed in the other interactions, but it is a source of some model error.

We believe that our second goal has also been met: valid comparisons of what is modeled can be made between the protocols. The Ethernet case suggests that care must be taken when transaction time is small (say, less than 50ms), but the performance of wide-area HTTP exchanges is dominated by network protocol behavior described in our model. Since the models capture the essence of performance in such networks, comparisons between the models should correspond to comparisons between the protocols operating in actual networks.

## 5.5 Additional Validation

A recent technical note by the World-Wide Web Consortium has suggested that pipelining substantially reduces packet counts for HTTP/1.1 [14]. We call the resulting protocol persistent-connection HTTP with pipelining, abbreviated HTTP/1.1+P. A comparison of their results with our model's predictions is particularly interesting both because their observations are made with different client and server software and because they have optimized the buffering of their system to improve performance. This comparison provides an additional level of validation of our model.

Their experiments compare first-fetch and the cache validation of a 42KB web page with 41 embedded images totalling 125KB. They examined performance for three networks: high bandwidth, low latency; high bandwidth, high latency; and low bandwidth, high latency. These nearly match our Ethernet, Fast-Internet, and Modem results and are shown in Table 2 as N-Ethernet, N-Fast-Internet, and N-Modem, although each case used a 1460B *mss*. (We estimated bandwidth for N-Fast-Internet based on examination of their traces; they presented bandwidths and *rtt*s for the other cases.) They considered four protocols: HTTP/1.0 with multiple parallel connections, HTTP/1.1, HTTP/1.1 with pipelining, and HTTP/1.1 with pipelining and compression. We consider only the case of HTTP/1.1 with pipelining, designating it HTTP/1.1+P. (We did not consider HTTP/1.0 because we do not model parallel connections, HTTP/1.1 because of the buffering problems they experienced, and HTTP/1.1 with pipelining and compression because we do not model compression.) Their client software was either a custom robot in the cases we consider. Their server software was either Apache or Jigsaw. A complete description of their methodology can be found in their technical note [14].

Table 8 summarizes the results of their measurements and our predictions. The adjusted portion of the prediction corresponds to addition of a 3.7ms server processing time.

The N-Ethernet and N-Fast-Internet cases show substantial discrepancy from our predicted values. We do not have enough information about their traces to understand the discrepancy for the N-

| | | | | prediction | | | ratio |
|---|---|---|---|---|---|---|---|
| **protocol** | **implement.** | **network** | **server** | **basic** | **adjusted** | **measurement** | **m:a** |
| caching-TCP | HTTP/1.1+P | N-Ethernet | Jigsaw | 160ms | 316ms | 690ms | 2.18 |
| caching-TCP | HTTP/1.1+P | N-Ethernet | Apache | 160 | 316 | 520 | 1.64 |
| caching-TCP | HTTP/1.1+P | N-Fast-Internet | Jigsaw | 1470 | 1620 | 2860 / 1860 | 1.77 / 1.15 |
| caching-TCP | HTTP/1.1+P | N-Fast-Internet | Apache | 1470 | 1620 | 3500 / 2419 | 2.16 / 1.49 |
| caching-TCP | HTTP/1.1+P | N-Modem | Jigsaw | 49600 | 49800 | 52810 | 1.06 |
| caching-TCP | HTTP/1.1+P | N-Modem | Apache | 49600 | 49800 | 52360 | 1.05 |

**Table 8: Additional validation experiments for our models.** These experiments use the workload described in Section 5.5. *Basic* indicates our basic (unadjusted) model, *adjusted* is the model adjusted for processing time, *measurement* indicates performance as measured in [14] with two values for N-Fast-Internet as described in Section 5.5, *ratio m:a* shows the ratio of the measurement to prediction/adjusted.

Ethernet case at this time, although, as described in Section 5.4, in LANs, per-packet processing (which is not considered in our model) can overwhelm connection startup costs.

For the N-Fast-Internet case we also found substantial discrepancy (1.77–2.16 times slower performance than predicted). Examination of their traces for this network configuration shows a consistent stall of about 1 second following the third segment of the reply. We believe that this stall is due to an interaction between a short TCP segment and TCP silly-window avoidance [22] similar to the odd/short-final-segment problem we encountered in our experiments [21]. If so, this interaction can be avoided by appropriate buffering. We correct for it by subtracting 1 second from the measured times. With this correction our model is much closer to the measured values which are 1.15–1.49 times slower.

For the N-Modem case the prediction corresponds closely to observed performance. These experiments corroborate our validation, suggesting that although our models can be inaccurate when applied to LANs, they can provide guidance to protocol designers for wide-area and low-bandwidth network conditions. We also note that models can provide a useful "sanity check" against observed performance and led us to investigate the anomaly in the N-Fast-Internet case.

# 6 Protocol Discussion

We have presented analytic models for HTTP over several transport protocols and demonstrated that, with care, these models can be applied to current as well as future network characteristics. From this work we can draw several conclusions about the interactions between HTTP and different transport protocols.

First, HTTP over TCP overhead is fairly low under networking characteristics today. Figure 4 shows a contour plot of TCP overhead for various network characteristics for the small-cluster workload. In this two-dimensional representation of the graph of Figure 2 we solve for overhead ($S_{TCP}/S_{min}$) for a set of representative points and employ linear interpolation between them. We show contour lines at every 50% increase in overhead. Of networking technologies deployed today, only the Fast-Internet case shows substantial overhead. Modem and ISDN technologies used for the "last mile" of the Internet today show moderate overhead when coupled with wide-area latency, but little overhead if the server is nearby.

Second, TCP overhead becomes significant when the bandwidth–delay product rises. Again referring to Figure 4, the fast-Internet performance shows substantial room for improvement (current performance is 5.20 times slower than the theoretically minimal transfer time), as do developing last-mile technologies such as ADSL and DirecPC[12].

[12] The DirecPC region falls out of scale and is not shown in

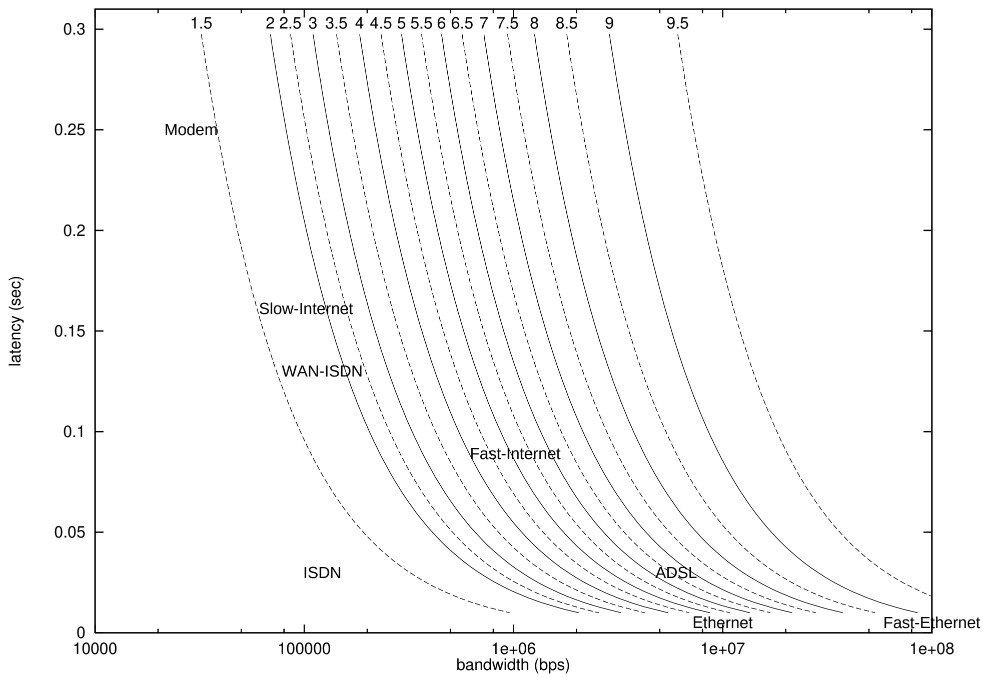


**Figure 4: Predicted overheads of HTTP over TCP relative to minimum possible costs.** The two axes show primary network parameters *bw* and *rtt*; segment size and workload are fixed at 512 and small-cluster, respectively. Contour lines show the surface corresponding to TCP overhead ($S_{TCP}/S_{min}$); solid lines are a factor of 1, dashed lines 0.5 times minimum. Label centers indicate bandwidth and latency points corresponding to the sample networks described in the text.

In these cases HTTP optimizations become important. Figure 5 shows the advantage of connection-caching protocols in different network configurations. In this graph, the long dashed line shows when standard HTTP takes 1.5 times as long as caching protocols, while dotted lines show intervals of 0.1. As can be seen, performance is marginally better in many cases (Modem, ISDN, and Slow-Internet); caching protocols are 80% faster only when presented with network characteristics similar to Fast-Internet (moderate bandwidth and latency), ADSL (high-bandwidth, low-latency), or DirecPC (high bandwidth and latency). The performance improvement always approaches a workload-dependent limit as the bandwidth–delay product rises; in this case the asymptote is 2, the ratio of 8:4 (non-caching:caching) round-trip delays.

A recent technical note by W3C has suggested that pipelining substantially reduces packet counts for persistent-connection HTTP [14]. Although they substantially reduce packet counts, their measurements of elapsed times support the conclusion that HTTP over caching-TCP protocols offer comparatively modest performance improvements over low-bandwidth–delay connections today but can provide substantial improvement when conditions approach the Fast-Internet case.

We note that our model can be used to predict HTTP performance for network technologies only now being deployed such as ADSL and DirecPC. The ability to vary workload and network characteristics is important here.

Finally, our protocol analysis has influenced design of UDP-based protocols at ISI. We are currently in the process of adapting ARDP to use TCP-like congestion avoidance algorithms. As a result of this study we have concluded that ARDP must cache information about recent congestion window behavior to provide good performance for large request–response exchanges.

the graph.

## 7 Conclusions and Future Work

This work makes three contributions to the study of HTTP. First, we have developed a simple analytic model for HTTP performance over different networks and transport protocols. Second, we have used this model to compare the relative performance of existing protocols for various network characteristics and workloads. Finally, this model has given us insight into the needs of request–response-style protocols.

Our analytic model of request–response performance is important both because it allows comparison of existing protocols under current network characteristics and because it allows prediction of protocol performance under future networking and workload characteristics. Our model predicts web performance within 5% of measured values for wide-area traffic. For networks with high bandwidth and low delay it becomes less accurate as non-modeled costs become noticeable. With this caveat, we believe that the model can be an effective means of comparing different protocols at a given network configuration and across different network characteristics.

In addition to providing a model useful for HTTP, our analysis of slow-start behavior applies to other uses of TCP where transient behavior cannot be ignored. Applications might include RPC-systems, and transfer of short e-mail messages or FTP of short files.

By applying our model to existing protocols and networks we were able to draw several conclusions about their behavior. We confirmed that TCP overhead is low when the bandwidth–delay product is low. In the Ethernet, modem, and ISDN cases overhead was consistently less than 25% for our workloads. Even when latency rose to WAN levels, modem and ISDN overhead was only moderate for certain workloads. We demonstrated that overhead was very significant when the bandwidth–delay product was large.

Connection caching protocols reduce overhead for the cluster cases (where a cluster represents the text and images that make up a single web page); we therefore conclude that these protocols will be useful even if users visit only single "pages" on sites before

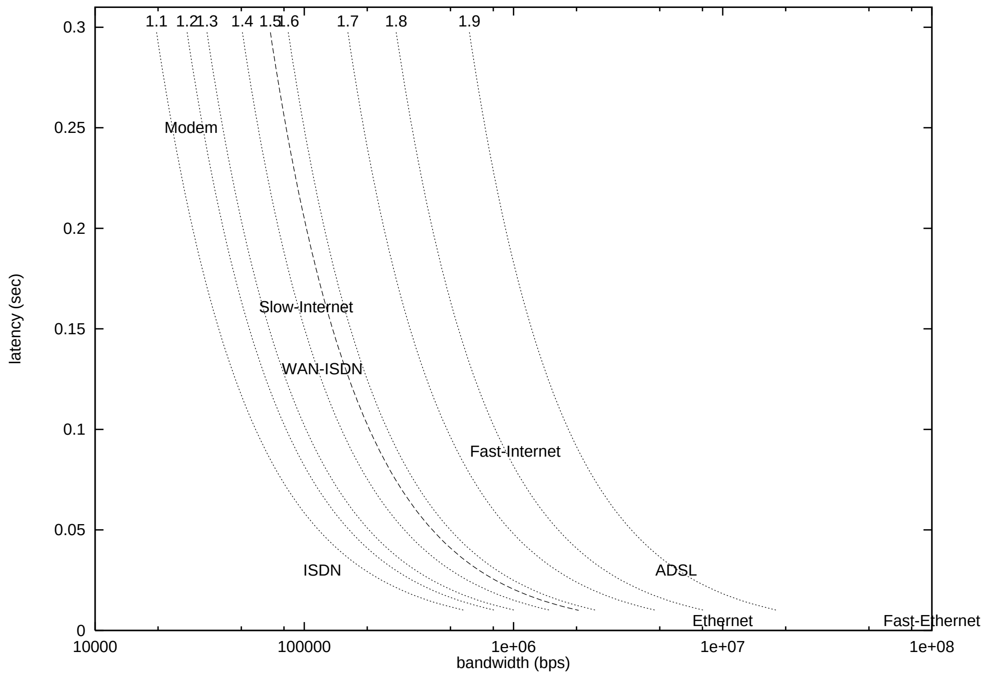


**Figure 5: Predicted ratio of HTTP over TCP to HTTP over caching TCP, assuming no initial connection caching.** The two axes show primary network parameters *bw* and *rtt*; segment size and workload are fixed at 512 and small-cluster, respectively. Contour lines show the surface corresponding to the ratio $S_{TCP}/S_{\mathit{first\text{-}miss}}$; long dashed lines are a factor of 0.5; fine dashed lines, 0.1. Label centers indicate bandwidth and latency points corresponding to the sample networks described in the text.

changing servers.

Finally, validation of our model has led to insight into request–response protocol design and suggested several areas for future work. Validation of these experiments have detected interactions between application- and kernel-level networking that substantially reduce performance [21].

A broader question is how to optimize TCP for brief, request–response-style traffic. We are currently exploring two approaches to this problem. We are examining how TCP congestion-control information should be initialized for multiple connections separated by space or time [23]; this work investigates alternatives to divide bandwidth among existing and new connections and for reusing cached congestion information. Given a large initial window, we are investigating how a rate-limited addition to slow-start can prevent overloading intermediate routers [21].

We have generalized our experiences with TCP to other transport protocols. We have also found that the performance of protocols that fail to cache congestion-control information suffers in high-bandwidth–delay conditions, and have modified our design for ARDP accordingly.

## Acknowledgments

We would like to thank Kedar Jog for his early work on our HTTP benchmarking scripts. The authors would like to thank Ted Faber for his discussions about web performance analysis, and B. Clifford Neuman, Rod Van Meter, Steven Augart, Brian Tung, Geoff Kuenning, Joseph Bannister, Jon Postel, and the anonymous referees for comments about the paper. Finally, we are grateful to Allison Mankin, Dante DeLucia, and B. Clifford Neuman for access to computer facilities for our cross-Internet measurements.

## Software Availability

We plan to make the software used in the validation of these measurements available, including the HTTP benchmarking programs and the programs used to evaluate the models. Please contact the authors for details, or watch ⟨`http://www.isi.edu/lsam/`⟩.

## References

[1] T. Berners-Lee, R. Cailliae, A. Luotonen, H. F. Nielsen, and A. Secret, "The World-Wide Web," *Communications of the ACM*, vol. 37, pp. 76–82, Aug. 1994.

[2] V. Paxson, "Empirically-derived analytic models of wide-area TCP connections," *ACM/IEEE Transactions on Networking*, vol. 2, pp. 316–336, Aug. 1994.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol—HTTP/1.0," RFC 1945, Internet Request For Comments, May 1995.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol—HTTP/1.1," RFC 2068, Internet Request For Comments, Jan. 1997.

[5] V. Jacobson, "Congestion avoidance and control," in *Proceedings of the SIGCOMM '88*, pp. 314–329, ACM, Aug. 1988.

[6] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW client-based traces," Tech. Rep. 95-010, Boston University, Apr. 1995.

[7] J. Touch, "Defining 'high speed' protocols : Five challenges and an example that survives the challenges," *IEEE Journal of Selected Areas in Communication*, vol. 13, pp. 828–835, June 1995.

[8] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," in *Proceedings of the ACM SIGMETRICS*, pp. 126–137, ACM, May 1996.

[9] R. Braden, "Extending TCP for transactions—concepts," RFC 1379, Internet Request For Comments, Nov. 1992.

[10] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," in *Proceedings of the Second International World Wide Web Conference*, Oct. 1994.

[11] J. Touch, J. Heidemann, and K. Obraczka, "Analysis of HTTP performance." Released as web page `http://www.isi.edu/lsam/publications-/http-perf/`, Currently submitted for publication, June 1996.

[12] J. C. Mogul, "The case for persistent-connection HTTP," in *Proceedings of the SIGCOMM '95*, pp. 299–313, ACM, Aug. 1995.

[13] S. E. Spero, "Analysis of HTTP performance problems." http://sunsite.unc.edu/mdma-release/http-prob.html, 1995.

[14] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG." NOTE-pipelining-970207, available as web page `http://www.w3.org/pub/WWW/-Protocols/HTTP/Performance/Pipeline.html`, 7 February 1997.

[15] R. Braden, "T/TCP—TCP extensions for transactions functional specification," RFC 1644, Internet Request For Comments, July 1994.

[16] W. R. Stevens, *TCP/IP Illustrated*, vol. 3. Addison-Wesley, 1996.

[17] B. C. Neuman, *The Virtual System Model: A Scalable Approach to Organizing Large Systems.* Ph.D. dissertation, University of Washington, 1992.

[18] D. DeLucia, "Direcpc performance." Personal communication., Oct. 1996.

[19] R. Braden, "Requirements for Internet hosts—communication layers," RFC 1122, Internet Request For Comments, Oct. 1989.

[20] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, Internet Request For Comments, Nov. 1990.

[21] J. Heidemann, "Performance interactions between P-HTTP and TCP implementations," *ACM Computer Communication Review*, vol. 27, pp. 65–73, Apr. 1997.

[22] D. D. Clark, "Window and acknowlegement strategy in TCP," RFC 813, Internet Request For Comments, July 1982.

[23] J. Touch, "TCP control block interdependence," RFC 2140, Internet Request For Comments, Apr. 1997.

[24] S. Shenker, L. Zhang, and D. D. Clark, "Some observations on the dynamics of a congestion control algorithm," *ACM Computer Communication Review*, vol. 20, pp. 30–39, Oct. 1990.

# A The TCP Slow-Start Algorithm in Detail

As described in Section 4.3.1, the TCP slow-start algorithm limits transmission by congestion window (*cwnd*) when a connection begins. Table 5 summarizes our analysis of slow-start performance. This appendix looks at the details behind this table, both the rate at which the congestion window opens and the amount of time spent waiting.

## A.1 The slow-start rate

The basic slow-start algorithm (as presented in Jacobson [5]) is that the *cwnd* begins at one segment worth of data and then is increased by an additional segment for each ACK received. This algorithm results in an exponential increase in *cwnd*; when *cwnd* reaches a threshold (*ssthresh*, initialized to 64KB), this increase is slowed to linear ($1/cwnd$ per ACK received). The exact rate of this exponential is dependent on the receiver's acknowledgment rate and will be bounded by *muws*. In this appendix we assume infinite *muws* and *ssthresh* and examine the effect of different acknowledgment rates. We also continue with the assumptions used in the rest of the paper: connection bandwidth and *rtt* are stable over the length of the connection and packet loss does not occur. In a real system, *cwnd* growth will be limited by packet loss due to congestion, buffer overflow, or connection window size.

We can therefore divide TCP behavior into a period consisting of a series of transmitted segments followed by a *stall*.[13] Formally, we define $segs(i)$ to be the number of segments sent in the $i^{\text{th}}$ period. To derive $segs(i)$ we will use $cwnd(i)$, the congestion window at the beginning of the period (measured in segments), $acks(i)$, the number of acknowledgment messages sent in response to $segs(i)$, and $unacked(i)$, the number of unacknowledged segments in period $i$. The number of segments sent in period $i$ is given by the following recurrence relation:

[13] Since segments are sent only in response to an ACK, segments tend to be sent back-to-back (this behavior was first noted by Shenker, Zhang and Clark in simulation experiments [24]).

$$segs(i) = cwnd(i) - unacked(i) \tag{17}$$

where,

$$cwnd(i) = cwnd(i-1) + acks(i-1) \tag{18}$$
$$cwnd(1) = 2$$

Our goal is to determine how many stalls occur when sending a given number of packets. The cumulative number of segments sent is helpful:

$$csegs(i) = \sum_{i=1}^{n} segs(i)$$

These formulae specify the sender-side aspects of slow-start. The receiver influences slow-start by its ACK rate. We will indicate the client's ACK policy with subscripts. For a client that acknowledges each packet,

$$\begin{aligned} acks_{ae}(i) &= segs_{ae}(i) \\ unacked_{ae}(i) &= 0 \end{aligned}$$

so from Equations 17 and 18,

$$\begin{aligned} segs_{ae}(i) &= cwnd_{ae}(i) \\ cwnd_{ae}(i) &= cwnd_{ae}(i-1) + acks_{ae}(i-1) \\ cwnd_{ae}(1) &= 2 \end{aligned}$$

The recurrence relation for $segs_{ae}(i)$ simplifies to the familiar exponential:

$$segs_{ae}(i) = 2^i$$

The fourth column of Table 5 shows sample values of $segs_{ae}(i)$ and $csegs_{ae}(i)$, the cumulative number of segments sent when clients acknowledge every segment.

These equations describe TCP behavior for older (4.3BSD-Tahoe) implementations; modern implementations implement delayed acknowledgments [19].

## A.2 Delayed Acknowledgments

TCP implementations with delayed ACKs send ACKs only after receipt of two full-size segments or a delay of up to a half-second. (Most BSD-derived implementations limit this delay to 200ms.) This approach avoids ACKs for many short segments while preserving TCP ACK-clocking. It risks transient effects in data-stream start-up [21], and it also reduces the rate of *cwnd* growth.

We can place a lower-bound on the ACK rate by assuming that delayed ACKs never occur (or that they occur only when all segments have been acknowledged). If we assume that delayed ACKs were timed from receipt of the last (odd) packet, and if the *rtt* was less than the delay, then delayed acknowledgments will never trigger. We adjust for this in our recurrence by halving the number of ACKs per stall, rounding and carrying over appropriately:

$$\begin{aligned} acks_{nda}(i) &= \left\lfloor \frac{segs_{nda}(i) + unacked_{nda}(i-1)}{2} \right\rfloor \\ unacked_{nda}(0) &= 0 \\ unacked_{nda}(i) &= segs_{nda}(i) + unacked_{nda}(i-1) \\ &\quad - acks_{nda}(i) \times 2 \end{aligned}$$

Again, from Equations 17 and 18,

$$\begin{aligned} segs_{nda}(i) &= cwnd_{nda}(i) - unacked_{nda}(i) \\ cwnd_{nda}(i) &= cwnd_{nda}(i-1) + acks_{nda}(i-1) \\ cwnd_{nda}(1) &= 2 \end{aligned}$$

The effects of this algorithm on slow-start performance are illustrated in the second column of Table 5, with $csegs_{nda}(i)$ shown in parentheses. Although both $csegs_{ae}(i)$ and $csegs_{nda}(i)$ grow exponentially, $csegs_{nda}(i)$ lags substantially.

This description is slightly more pessimistic than actually occurs in Berkeley TCP implementations. In BSD the delayed ACK timer fires independent of segment receipt every 200ms, so we expect delayed ACKs to be generated occasionally, each time acknowledging a single segment.

We can place an upper bound on Reno's performance by assuming that the delayed-ACK timer always fires immediately for the last odd-numbered segment of any stall. This means that the receiver acknowledges every other packet and delay-acknowledges odd packets. The revised relations are:

$$\begin{aligned} acks_{da}(i) &= \left\lceil \frac{segs_{da}(i)}{2} \right\rceil \\ unacked_{da}(i) &= 0 \end{aligned}$$

And, from Equations 17 and 18,

$$\begin{aligned} segs_{da}(i) &= cwnd_{da}(i) \\ cwnd_{da}(i) &= cwnd_{da}(i-1) + acks_{da}(i-1) \\ cwnd_{da}(1) &= 2 \end{aligned}$$

Both $segs_{da}(i)$ and $csegs_{da}(i)$ are shown in the third column of Table 5. While $csegs_{da}(i)$ is somewhat larger than $csegs_{nda}(i)$, it is still much lower than $csegs_{ae}(i)$.

## A.3 Amount of Wasted Time

We would like to quantify the amount of time wasted during each stall. An upper bound on wasted time is one *rtt* per stall: the time an ACK takes to return to the server and its replacement segment to travel to the client. A more accurate estimate would consider that the ACK which triggers the start of the next period is generated by the first one or two segments of the current period. Following the ACK, therefore, the client spends time usefully receiving any other segments of the first period. We can see this in Figure 1; in the first stall, the second segment is received after the first ACK has been sent. (The client here must implement the ACK-every-segment policy.)

We can quantify the amount of useful work accomplished during a stall, and from there the exact amount of wasted time:

$$\begin{aligned} usefulstalltime(i) &= (segs(i) - k)/bw \\ wastedstalltime(i) &= rtt - usefulstalltime(i) \end{aligned}$$

where $k$ is either 1 (if every segment is acknowledged) or 2 (if delayed acknowledgments are used).

To determine the amount of wasted time for an entire transaction we must know the number of stalls that occur over transaction. Let *ss_segs* be the number of segments sent while TCP slow-starts. A connection will slow-start until it either runs out of data to send, reaches *muws* (and so is sending data continuously), or reaches *ssthresh* and therefore begins congestion avoidance. Thus:

$$ss\_segs = \min(muws, reply_{size}/mss, ssthresh)$$

The number of stalls across *ss_segs*, *stalls*(*ss_segs*) is then the smallest $n$ such that:

$$ss\_segs \leq csegs(i)$$

For a given *ss_segs*, *stalls*(*ss_segs*) can be obtained from Table 5's first column using the appropriate receiver's ACK algorithm. Finally, we can determine the slow-start delay for a transaction:

$$slowstart_{TCP} = \sum_{i=1}^{stalls(ss\_segs)} wastedstalltime(i)$$