Modeling the Performance of Short TCP Connections

Neal Cardwell, Stefan Savage, Tom Anderson

Department of Computer Science and Engineering
University of Washington

October, 1998

Abstract

Recently researchers have proposed several analytic models of TCP performance. Several of these models accurately describe the steady-state behavior of long TCP connections transferring megabytes of data. Several address the performance of short TCP connections transferring a few kilobytes of data, but assume no packet loss. However, it is not obvious how applicable these models are, since a wealth of evidence suggests that in the most popular use of TCP today – HTTP, the protocol of the World-Wide Web – connections are short, often around 10KB, and often suffer high packet loss rates in the neighborhood of 5%.

This paper examines how well these existing models capture the behavior of short TCP connections under realistic loss rates. In addition, we describe two new models for TCP performance: a model for short TCP connections that experience no packet loss, and a generalization of this model for short connections that do experience packet loss. Comparing these models against simulated TCP flows, we find that both the new models and, somewhat surprisingly, the previously-proposed steady-state and lossless short flow models closely reflect TCP performance under a wide variety of simulated conditions. However, several experiments suggest that the new short flow models may provide a much closer fit for the measured performance of short TCP flows in the Internet.

1 Introduction

TCP is the reliable transport protocol used for many of the Internet's most popular applications: e-mail, news, remote login, file transfer, some streaming audio and video protocols, and the web. Because these applications are the dominant applications on the Internet today, TCP controls the vast majority of traffic on the Internet – about 95% of the bytes and 90% of the packets sent over the Internet, according to one recent study [TMW97].

In addition to ensuring that all the data sent from the source endpoint of a TCP connection (or "flow") eventually reaches its destination endpoint, TCP is responsible for deciding when and how fast to send data. Because millions of users are sharing the Internet at any particular time, TCP must constantly adapt to the available bandwidth, slowing down when it detects congestion, and speeding up when it deduces that its fair share of the bandwidth has increased. This is a hard task, because the Internet provides only a few clues as to the correct rate: whether packets are reaching their destination, and how long it takes them. As a result, over the last fifteen years TCP has grown increasingly complex; the most popular exposition of the reference implementation is more than 1100 pages long [Ste95]!

Despite this daunting complexity, recently several research efforts have proposed mathematical models for TCP performance based on the characteristics of the network path and the TCP implementation itself.

Analytic models for TCP performance have many uses:

- To improve TCP, we need to understanding why TCP performs as it does, how efficient it is, and what limits its performance - the network, the protocol specification, a particular TCP implementation, or a particular application. For example, there are several current proposals for changing TCP that we can evaluate analytically: increasing TCP's initial window size [SAP98, AFP98, PN98, SP98, AHO98], sharing information about network path characteristics between TCP connections [Tou97, BPS+98, SSK97], changes in the way TCP increases its sending rate in response to acknowledgments [HSMK98, All98], and new TCP-like transport protocols [GCMW98].
- To improve networks or make better routing decisions in existing networks, we must understand the relative importance of network parameters, and what their trade-offs are (loss rate vs latency, for example).

- Simulations and analytic models of new applications and algorithms, such as HTTP 1.1 [FGM+97] or web caching, can use models of TCP performance to predict the performance that applications can expect from TCP.
- Researchers designing new network transport protocols - for disseminating unicast or multicast real-time audio or video, for example - need to have a model of how TCP behaves in order to design TCP-friendly algorithms that adapt to perceived congestion in a manner that will be fair to TCP flows competing for bandwidth along the same network path.

Unfortunately, as useful as analytic models are, most of the currently-proposed analytic models assume arbitrarily-long TCP connections, where the complex, random behavior of TCP can be summarized accurately by deriving the expected bandwidth in the steady state. However, a wealth of evidence suggests that most TCP connections today are short - usually transferring under 10KB [CBC95, BSSK97, Mah97a, GB97, TMW97]. Those models that consider shorter TCP connections assume that there are no lost packets. This is an unrealistic assumption in an Internet where loss rates are often around 5% [Pax97, BPS+98].

Hence it is important to understand how well the steady-state, lossy models and short-flow, loss-less models model actual TCP behavior in networks with realistic packet loss patterns and representative transfer sizes.

The aim of this paper is to use analytic models, simulation, and measurement to understand how well several TCP performance models fit TCP behavior in scenarios representative of today's Internet. In addition, we propose an analytic model that appears to more accurately fit TCP performance in the case of short transfers over network paths with low or high loss.

The paper is organized as follows. Section 2 reviews the mechanics of TCP behavior, outlines results in modeling this behavior, and summarizes measurement results that diverge from the assumptions in these models. Section 3 describes a model of short TCP flows that do not suffer from packet losses, and Section 4 describes a model for short flows that incur random losses. We compare these two models with steady-state, lossy models using simulation in Section 5 and measurement of actual TCP implementations transferring data over the Internet in Section 6. Section 7 describes the limitations with this approach, and Section 8 summarizes our conclusions.

2 Background

2.1 The Mechanics of TCP

TCP provides a reliable, ordered, bi-directional byte stream between two applications on any hosts in the Internet. Furthermore, TCP provides flow control, so that the sender does not send faster than the receiver is prepared to receive. In addition, TCP provides congestion avoidance and control, so that the sender avoids sending fast enough to cause network congestion, and adapts to any congestion it detects by sending slower [Jac88, Ste94, Ste97].

To implement this service, TCP gives each byte in the stream a unique 32-bit sequence number. Then it divides the sequence of bytes into segments of some maximum segment size, or MSS. The MSS is typically 536 or 1460 bytes [Ste96, TMW97]. Each TCP segment is placed in an IP packet and then sent to the receiver.

Before sending any data, the two endpoints must establish a connection between themselves by a three-way handshake (see figure 1). First, the connection initiator sends a SYN ("synchronize" sequence numbers) segment with its initial sequence number to the other side. Then the other side responds by sending a segment with an acknowledgment (ACK) for this sequence number and a starting sequence number of its own. Finally, the initiator sends an ACK segment acknowledging the other side's initial sequence number.

Once the connection has been established, the sender begins sending data segments. When the receiver receives a data segment, it send back an acknowledgment that specifies the highest in-sequence sequence number that it has received. The sender then knows that every byte up to that sequence number has been received. For example, if a receiver receives segments 1 and 2 and 4, it will acknowledge segment 2, since there is a hole where segment 3 should be. Receivers do not always acknowledge segments immediately after receiving them. TCP implementations vary widely in this regard, but they are supposed to use a "delayed ACK" scheme so that they can coalesce acknowledgments for up to two segments.

As the sender is sending data segments, it must decide when and how fast to send them. For this, TCP uses a window-based scheme, with a receiver-advertised window, wnd, and a congestion window, $cwnd$; the effective window is the minimum of these two windows. The TCP sender always sends as much data as its effective window allows, and then waits for an acknowledgment. When it receives an acknowledgment it sets wnd to the window advertised in this ACK, increases $cwnd$, and again sends as much data as its effective window allows.

Because TCP sends one window's worth of data, waits one round-trip time (RTT) for acknowledgments, and then recomputes its windows, the average throughput of a TCP connection is equal to $min(cwnd, wnd)/RTT$. In a well-designed TCP implementation, the receiver window is fixed at 64KB, so for typical networks the sending rate is completely determined by the evolution of $cwnd$ over time. There are two modes in which $cwnd$ evolves: slow start, and congestion avoidance.

In slow start, $cwnd$ starts at some minimum window, typically one MSS, and increases by one MSS for each acknowledgment segment received. With delayed acknowledgments, every other data segment elicits an ACK, so typically a $cwnd$ of k packets results in $k/2$ acks. This, in turn, results in a new $cwnd$ of $k + k/2 = 1.5 \cdot k$. Thus, during slow start the $cwnd$ typically grows exponentially by factors of 1.5. Because slow start begins with such a small window and rapidly increases the rate at which it sends, TCP uses it whenever it is not sure how fast it should send: at the beginning of a TCP connection, after idle periods, and after extreme congestion. Figure 1 shows an example of slow start behavior.

TCP enters the more conservative congestion avoidance mode once it thinks it has reached a safe rate. During this mode, TCP increases its $cwnd$ by $MSS \cdot MSS/cwnd$ for every acknowledgment it receives. Since, with delayed acknowledgments, k segments should typically receive $k/2$ acknowledgments, each RTT TCP will make $\frac{1}{2} \cdot \frac{cwnd}{MSS}$ increases, each approximately $MSS \cdot MSS/cwnd$ bytes, for a total increase in $cwnd$ of $\frac{1}{2} \cdot MSS$ per RTT. That is, during congestion avoidance, TCP slowly additively increases $cwnd$ by one half segment per RTT. This is intended to be fast enough that TCP will eventually avail itself of any new bandwidth that becomes available, but slow enough that it will not cause congestion (in fact, it does not always avoid congestion well [BOP94]).

The final important aspect of TCP is its means of detecting lost packets and congestion. A packet loss has two implications for TCP. First, because TCP endeavors to provide a reliable transport service, it must retransmit the lost segment until the receiver acknowledges that it has successfully received the packet. Second, in today's Internet, packet loss is usually a signal of congestion. Internet congestion results when packets arrive at a router faster than they can leave, and this condition persists long enough to overflow the queues built to mitigate short bursts of rate mismatch. TCP uses two methods to detect packet loss and makes two corresponding responses.

The first method is retransmission timeout (RTO). Whenever TCP sends a segment, it sets a timer to go off at a point in the future when it expects that the acknowledgment for that packet will have arrived. This timer is set according to measurements of the round-trip time between when segments are sent and when their acknowledgments are received. If TCP receives an acknowledgment for the segment, it cancels the retransmission timer. On the other hand, if this timer expires and TCP has not received an acknowledgment for this segment, then it assumes that the segment was lost. When TCP detects a loss via this timeout mechanism, it retransmits the lost packet. Furthermore, TCP conservatively assumes that there may be severe congestion along this path, so it sets $cwnd$ to one MSS and begins sending again in slow start mode.

The second method is based on noting holes in the sequence numbers of segments received by the destination. Suppose a TCP sender sends segments 1, 2, 3, 4, 5, and 6, and then segment 3 is lost. The receiver will acknowledge packets 1 and 2. However, because TCP acknowledgments are cumulative, and because there was a hole where segment 3 should have been, when the receiver receives segments 4, 5, and 6 it can only acknowledge segment 2 once for each segment it receives beyond the hole. To handle scenarios such as this, when a TCP sender receives three or more duplicate acknowledgments for a segment, it assumes the packet was lost. Again, TCP starts by retransmitting the missing packet; this is referred to as a "fast retransmit," to contrast it with the retransmission timeouts, which are typically much slower. However, in these triple-duplicate ACK situations, because several segments past the lost packet successfully reached the destination, TCP assumes that the congestion is moderate. Hence, after the fast retransmission it halves its $cwnd$, rather than setting $cwnd$ back to one MSS.

There are several subtleties beyond this level, the most notable of which involve "fast recovery", where TCP uses a variety of heuristics to attempt to quickly recover from losses detected via triple duplicate ACKs. This is an active area of research [Hoe96, SAP98]. However, together, the connection establishment handshake, slow start, congestion avoidance, RTT estimates, retransmission timeouts, and fast retransmission largely determine the performance of TCP. By capturing these essential aspects of TCP, models can characterize the performance of TCP as a function of RTT, MSS, delayed ACK policy, RTO length, and loss rate.

2.2 Limitations of Current Models

A number of researchers have proposed models for TCP performance [Flo91, OKM96, HOT97, MSMO97, LM97, Kum98, PFTK98]. Most of them analyze the steady-state throughput of long bulk-transfer TCP connections [Flo91, OKM96, MSMO97, LM97, Kum98, PFTK98]. One [HOT97] analyzes the performance of HTTP over TCP in the case where there is no packet loss. However, there is a wealth of evidence suggesting that most TCP connections are short, and that packet loss rates are not negligible.

Most TCP connections today are carrying HTTP data [TMW97]. In generic HTTP 1.0, a web browser opens up a new TCP connection for each object embedded in a web page [BLFF96, Ste96]. Several studies have found that this results in TCP connections with a median size of 2-3KB, an average transfer size of 8-12KB, with the vast majority of connections transferring less than 10KB, [BC94, CBC95, Mog95, Ste96, BSSK97, Mah97a, GB97, TMW97]. The vast majority of objects are smaller than the average because of the heavy-tailed distribution of object sizes. These flows are quite short in TCP terms – with a typical MSS of 1460 bytes, 10KB transfers are 7 segments long.

HTTP 1.1 and some implementations of HTTP 1.0 have support for persistent connections, which allow a single TCP connection to be used to download several objects [FGM$^+$97, NGBS$^+$97]. Persistent connections support is available in recent versions of the most popular browser and server software, though we know of no measurements discerning the fraction of HTTP transactions that use this feature. As persistent HTTP connections become more widely deployed, the average size of TCP connections may grow considerably. On the other hand, flows may not be too much longer; even if all of the data on a page is sent in a single connection, [Mah97a] found that the average size of all of the data on a web page was 26-32KB.

If connections are short and therefore the relevance of steady-state models unproven, one might hope that instead we could adopt a simple model of short flows along the lines of the model proposed in [HOT97]. Unfortunately, this model assumes that packet loss is negligible, whereas [BPS$^+$98] and [Pax97] report that packet loss rates in the Internet are in the neighborhood of 5% and may be climbing. With this in mind, Section 3 extends the work of [HOT97] by describing a simple parameterized model of short flows that do not experience packet loss. Then Section 4 generalizes this model to include flows that experience packet loss. Section 5 compares these models against simulation results, and Section 6 compares them against measurements of TCP performance in the Internet.

3 Short TCP Flows: Modeling Optimal Performance

3.1 Deriving a Model

Consider a TCP connection where a small amount of data is transferred in one direction, as is typical for an HTTP transaction (see figure 1). We can derive a model for the optimal performance of this connection by considering the behavior of the connection under ideal conditions: TCP implementations with properly-tuned send and receive windows and a high-bandwidth, symmetric path between sender and receiver with no losses and no queuing delay. Here "high bandwidth" means that $(\textit{packet size})/(\textit{bandwidth}) \ll \textit{propagation delay}$. In practice, these conditions are often achieved during low-traffic periods between machines whose bottleneck link is 10Mb/s or greater. On the other hand, low-bandwidth links such as modems do not meet these criteria.

First, the optimal short TCP connection begins with a successful three-way handshake. This handshake will take approximately twice the propagation delay between sender and receiver, or one round-trip time, denoted RTT.

If the connection initiator (client) is sending the data, then the data transfer can begin immediately. On the other hand, if the server is sending the data, as in HTTP, then it must wait $RTT/2$ for the acknowledgment of its SYN+ACK segment, and possibly a specific request from the client as well – an HTTP GET request, for example.

As the transfer begins, the sender will be in slow start mode; the connection will experience the best possible performance if it experiences no losses and *remains* in slow start for the remainder of the connection. To understand the performance of slow start mode under best-case conditions, we consider its progress in terms of "rounds" whose duration is equal to one RTT.

During the first round, as the sender starts sending, its $cwnd$ starts at some fixed initial window size, w. The TCP specification [SAP98, Bra89, Ste97] requires that w is one MSS, though some implementations use 2 MSS, and there are proposals to use a window of up to 3 or 4 MSS, as long as the combined size of the segments does not exceed 4380 bytes [SAP98, AFP98, PN98, SP98, AHO98].

When the initial window of data reaches the destination, the destination will respond with some number of acknowledgment packets. The receiver may delay its acknowledgment(s); let the average delay for the first ACK be t_{delack}. This value varies between implementations, and some implementations use an adaptive scheme, but a value around $t_{delack} = 100ms$ is typical.

When the first acknowledgments reach the sending host at the end of the first round, the sending TCP responds according to the slow start specification and increases its $cwnd$ by one MSS for each acknowledgment packet received.

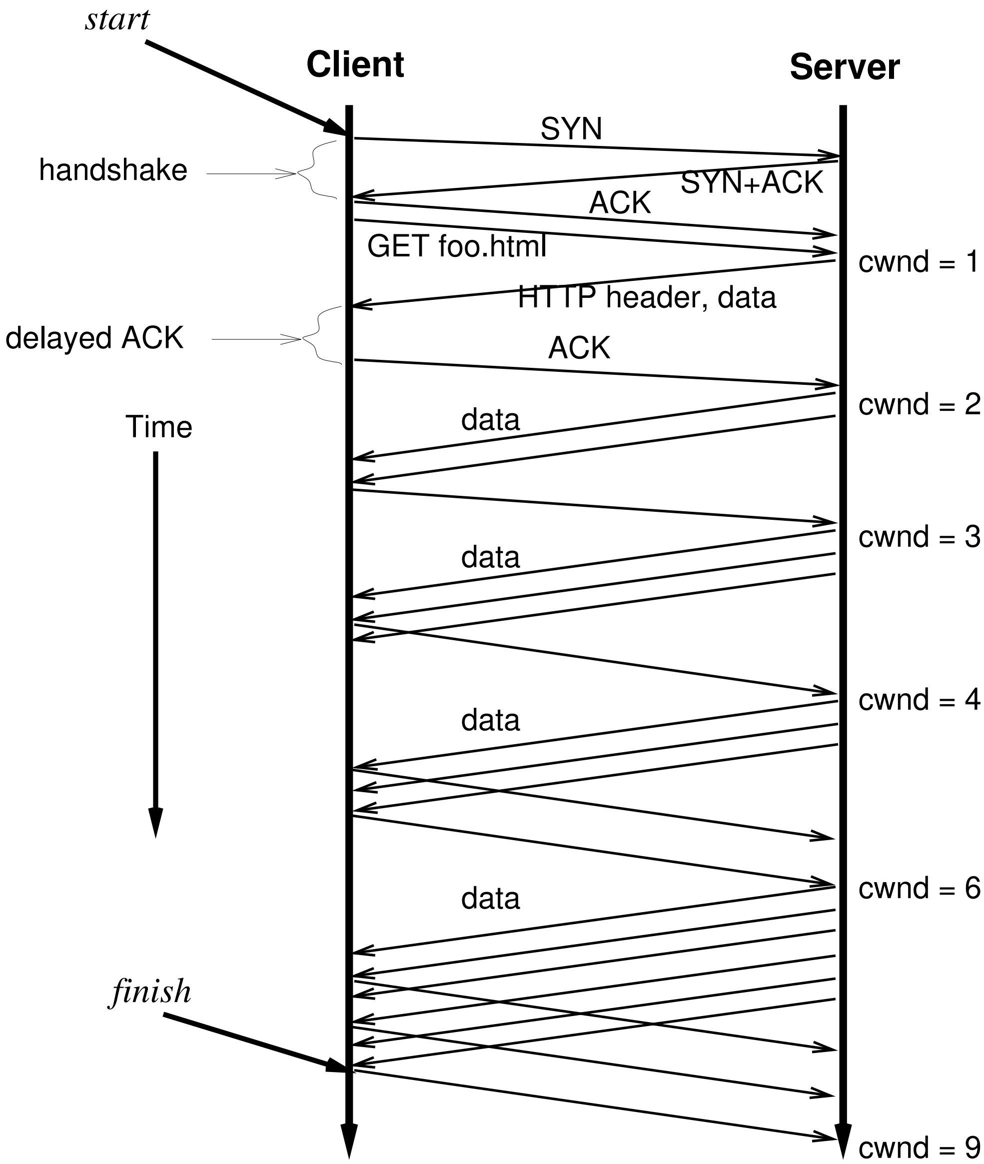

Figure 1: An example of an HTTP transaction over TCP. Note the delayed acknowledgments: the receiver generates a cumulative acknowledgment for every other data segment. When $cwnd = 4$, the sender can only send 3 packets because there is one unacknowledged packet. The connection tear-down is not pictured here, as it is not important to application performance.

Then the sender begins the second round by sending $cwnd$ packets. This process repeats until all the data reaches the destination host.

We can model this by letting $cwnd_i$ be the $cwnd$ of the sender at the beginning of round i, and $acks_i$ the number of acknowledgments sent by the receiver in round i. Since, in slow start, a sender increases its $cwnd$ by one MSS for each acknowledgment packet received, we have:

$$cwnd_{i+1} = cwnd_i + acks_i$$

The number of acknowledgments a receiver sends, $acks_i$, depends on the receiver implementation. A TCP receiver that implements delayed acknowledgments should send one acknowledgment for roughly every $b = 2$ packets [Bra89], though older implementations acknowledge every $(b = 1)$ packet, and a few newer research proposals are equivalent to acknowledging every packet [GCMW98, All98]. In either case,

$$acks_i = cwnd_i / b$$

and thus

$$cwnd_{i+1} = cwnd_i + cwnd_i / b = \left(1 + \frac{1}{b}\right) cwnd_i$$

This suggests that $cwnd_i$ can be modeled as a geometric series with ratio $r = 1 + \frac{1}{b}$. Thus $data_i$, the number of data segments sent in rounds $1 \ldots i$, is approximately

$$\begin{aligned} data_i &= \sum_{k=1}^{i} cwnd_k \\ &= w + w \cdot r + w \cdot r^2 + w \cdot r^3 + \cdots + w \cdot r^{i-1} \\ &= w \cdot \frac{r^i - 1}{r - 1} \end{aligned}$$

Solving for i, the number of slow start rounds to transfer $data_i$ segments of data, we arrive at:

$$i = \log_r \left(\frac{data_i (r-1)}{w} + 1 \right)$$

Finally, we can model the effect of opening k simultaneous TCP flows to evenly split the burden of transferring a total of $data$ bytes by letting each flow transfer $data/k$ bytes. This is similar to the approach that browsers take; most browsers open up four simultaneous TCP connections to download objects for a web page.

Thus the time required to open k TCP connections and transfer a total of $data$ bytes of data, incorporating the time for the handshake, delayed acknowledgment, and slow start from an initial window of w segments, is:

$$t = \log_r \left(\frac{data(r-1)}{w \cdot MSS \cdot k} + 1 \right) \cdot RTT + RTT + t_{delack} \qquad (1)$$

This generalizes the results from [HOT97, Mah97b]. [HOT97] goes through a similar process of reasoning for the specific case where $w = 1$, $r = 1.5$, and $k = 1$, but does not solve for a closed-form approximation. [Mah97b] solves for a closed-form approximation of this case, with a different result.

3.2 Lessons From the Optimal Model

First note that the delayed ACK penalty, although significant for short flows with small RTTs, only occurs in some combinations of sender and receiver implementations. Furthermore, it will probably disappear gradually over time, as TCP implementations adopt larger initial windows and adaptive delayed ACK strategies.

Next note that, neglecting the delayed ACK penalty and any computational overhead on the server or client, the transfer time for a short flow is linear in the RTT. This is reflected dramatically in the subjective user experience of web surfing - surfing at ISDN (128Kbps), T1 (1.5Mpbs), and Ethernet (10Mbps) speeds feels largely the same, since the latency for each technology is comparable. This dependence on RTT will make improving short TCP

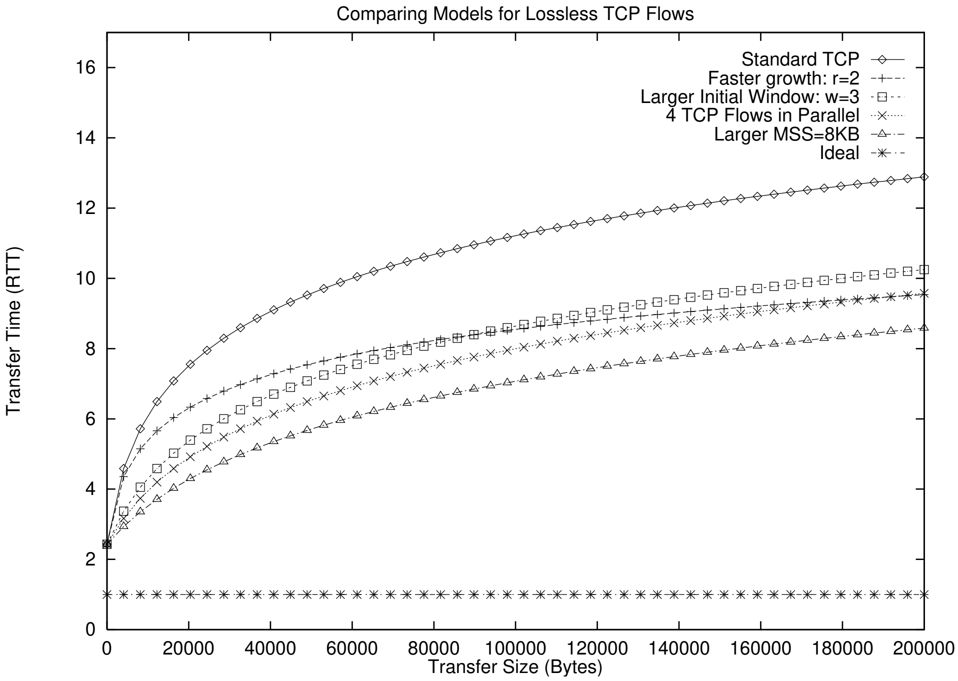

Figure 2: Comparing various implementations for short flows. Time is expressed in units of RTT. Except where noted, there is a single flow, with an MSS of 1460 bytes, $r = 1.5$, and $w = 1$. The delayed ACK penalty here is 1.43 RTT, equivalent to the cost of a 100ms delayed ack penalty for a transfer across the continental US.

flow performance challenging in the long run, since high-bandwidth networks are ensuring that RTT is increasingly determined simply by the speed-of-light delay between hosts.

Finally, note that, for reasonable (i.e. small) values of w and k, the transfer time is roughly logarithmic in the degree of the exponential growth.

Figure 2 compares several alternative TCP short flow implementations, as well as a lower bound on transfer time. This figure depicts the following alternatives:

- **Standard TCP:** This is TCP with a single flow, with an MSS of 1460 bytes, standard delayed ACKs yielding an exponential growth rate of $r = 1.5$, and a standard initial window of $w = 1$ segments.
- **Faster growth with $r = 2$:** Just like standard TCP except that the receiver ACKs each segment. This is equivalent to the slow start phase in TCP Tahoe or the WebTP proposal [GCMW98].
- **Larger initial window with $w = 3$:** Just like standard TCP, except with the proposed larger initial window of three segments.
- **Four TCP flows in parallel:** Just like standard TCP, except that there are four TCP flows in parallel, much as in a typical web browser.
- **Larger MSS of 9KB:** Just like standard TCP, except with an MSS of 9KB. This largely just to show the effect of changing MSS. Very few paths in the Internet allow packets bigger than 1500 bytes, the MTU of Ethernet and PPP. However, it is conceivable that at some future point some paths in the Internet may accept 9KB packets, if Gigabit Ethernet LANs using 9KB Jumbo Frames [jum98] become popular.
- **Ideal:** This case shows the fastest performance achievable with any request-response protocol. The time given is the latency for a client to transfer a request to a server ($RTT/2$), plus the latency for the server's response packets to propagate to the client ($RTT/2$). We assume the transmission time (data size divided by bottleneck bandwidth) is negligible. This level of performance could be achieved over an Internet path only if 1) the bottleneck bandwidth were relatively high – increasingly true for clients and servers on 10Mbps LANs and

whose connecting path has only T3 or better links – and 2) the sending TCP implementation knew somehow automagically that it had all of the bottleneck bandwidth to itself.

This figure shows that all the TCP alternatives share a similar logarithmic shape. The TCP alternatives with a larger initial window $w = 3$, $k = 4$ flows in parallel, and larger 9KB segments all have similar performance, performance consistently a few RTT better than standard TCP. In fact, in a lossless scenario, these parameters are equivalent in terms of performance, since an increase by a factor of c in any of them results in the TCP sender sending c times more data in each RTT. Their equivalent effect is evident in equation 1, where they sit together in the divisor $w \cdot MSS \cdot k$. Their effect is small because exponential growth quickly outstrips a constant factor improvement; in the model this is evident from the fact that they reside inside a $\log_r$.

In a real world scenario with packet losses, these three schemes are not actually equivalent. Using larger segments should decrease the loss rate, assuming routers queue some fixed number of packets of any size. Using a larger initial window should help the TCP sender recover from losses using fast retransmit and fast recovery more often, since $cwnd$ will grow quickly. And using k separate flows will help keep data flowing even if one of the flows suffers a loss.

Given that a larger initial window is the easiest of these to implement, these results suggest that this is an attractive alternative.

Not surprisingly, the most effective TCP implementation for long, lossless flows is the scheme that uses faster exponential growth, with $r = 2$.

Of course, the ideal protocol protocol, which sends data as fast as possible, is far faster than any of the other schemes. This is fundamentally because real-world TCP implementations must spend a few RTT figuring out the fastest possible safe rate. But before sending much closer to the actual available bandwidth, a real TCP implementation must have knowledge about the bandwidth that has been available recently. This suggests the advantage of preserving and sharing information about network conditions; this has been proposed in several different contexts [BSSK97, SSK97, Tou97, BPS+98, PK98, Hei97, SAA+98].

3.3 Limits of the Optimal Model

3.3.1 Bandwidth-Delay Product

This model of optimal performance of TCP flows assumes that the flow is able to send increasingly faster, without reaching the bottleneck bandwidth, or suffering any losses. However, even if we assume that the path is relatively high bandwidth and there are no random losses caused by other connections competing for bandwidth, a TCP flow that is long enough will eventually reach the bottleneck bandwidth. When this happens, the bandwidth achieved by the flow will be capped by the bottleneck bandwidth rate; if the transfer continues, the queue before the bottleneck link will build up, and if the flow continues still further the queue will fill up and the router will drop newly-arriving packets. Eventually the sender will detect the loss, leave slow start, and cut its sending rate.

How long do we expect a TCP flow to be able to continue in slow start before it reaches the bottleneck bandwidth and leaves the slow start growth regime? We expect that this will happen approximately when the rate at which it is sending is equal to the bottleneck bandwidth, B. Since TCP sends one window of data per RTT, this will happen when

$$\begin{aligned}
cwnd/RTT &= B \\
cwnd &= B \cdot RTT \\
w \cdot r^{i-1} &= B \cdot RTT \\
r^{i-1} &= \frac{B \cdot RTT}{w} \\
i - 1 &= log_r \left(\frac{B \cdot RTT}{w} \right) \\
i &= log_r \left(\frac{B \cdot RTT}{w} \right) + 1
\end{aligned}$$

By this time, the flow will have sent:

$$data_i = w \cdot \frac{r^i - 1}{r - 1}$$

$$= \quad w \cdot \frac{r^{log_r(\frac{B \cdot RTT}{w})+1} - 1}{r-1}$$
$$= \quad \frac{B \cdot RTT \cdot r - w}{r-1} \tag{2}$$

This means that, for a typical TCP implementation with $r = 1.5$ and $w = 1$, the TCP flow will transfer around three times the bandwidth-delay product before it fills the pipe and causes its own packet loss.

Bandwidth-delay products vary widely between different Internet paths. But we can consider two common cases: web surfing from home over a 28.8Kpbs modem and web surfing from work on an Ethernet LAN connected to the Internet via a 45Mbps T3 line. When surfing from home, US Internet users might expect to experience a bandwidth-delay product of $28,800$bits/sec $\times$ 0.700sec $=$ 2520bytes; in this scenario the model will not fit flows of longer than approximately 7.5KB. When surfing from a well-connected workplace or school, US Internet users might experience a bandwidth-delay product of 10Mb/sec $\times$ 0.070sec $=$ 87500bytes; in this scenario the model will not fit flows of longer than approximately 260KB. Given that TCP connections are about 10KB long and growing longer over time, this suggests that this model may be able to describe some connections over high bandwidth-delay paths, though not low bandwidth-delay paths.

3.3.2 Random Packet Loss

The other major limitation with this model, of course, is that it assumes a TCP flow suffers no losses, whereas TCP connections in the real Internet face random losses of both data segments and acknowledgments. However, given that TCP flows are often short, it is reasonable to expect that even in the face of random loss, many flows will not suffer any losses. For example, Figure 1 depicts a TCP flow with data packets, which could transfer 21KB of data, assuming a typical MSS of 1460 bytes. In this flow there are about 21 packets that must successfully reach the destination on the first try to ensure that performance is optimal. Assuming an i.i.d packet loss rate of 1%, there is about an 80% chance that such a flow will see optimal performance, and still a 34% chance with a 5% loss rate. To look at it another way, with an i.i.d. loss rate of 5%, we expect to be able to successfully transmit about 20 packets on average before encountering a loss; this is enough for a handshake and at least 9 segments of data and their ACKs, or about 13KB.

Since many flows over paths with loss rates of 1-10% will see optimal performance and many will not, it is critical for a model of TCP performance to capture both the optimal and degraded cases. Section 4 presents a model that attempts to do exactly this.

4 Modeling Short TCP Flows with Random Loss

The lossless model in equation 1 is based on the assumption that the flow in question suffers no packet losses. However, in the real world, packets often arrive at routers in bursts, where they are queued. Periodically a burst of packets fills up a queue and the router drops packets that arrive while this queue is full. When TCP flows suffer losses their $cwnd$ decreases, possibly after a long retransmission timeout (RTO). In either case, their behavior is no longer well-modeled by the lossless model.

Some packet losses experienced by a flow are caused by the flow's own actions. Section 3.3.1 described one case in which this is certainly true – when a flow exceeds the bottleneck link rate. Other times packet losses are largely independent of a flow's own actions, and result instead from the flow's packets encountering a queue that is full due to a random burst of packets from other flows traveling across that link. Short flows over a high-bandwidth path will usually not use a significant fraction of the path's bandwidth, and thus any queuing and resulting loss is usually a result of random bursts of traffic from other flows. As discussed in section 2.2, recent measurement results suggest that loss rates are often in the vicinity of 5%, and that loss rates are random, albeit dependent on a flow's own actions [Pax97].

Sections 4.1 and 4.2 describe two models for short TCP flows suffering from losses.

4.1 Adopting Steady-State Models

The most obvious approach is to adopt an accepted steady-state model and apply it to the case of short flows, hoping that perhaps it will happen to describe short flows well, even though they do not fit the assumptions of the steady-state model.

The steady-state model that is most promising as a candidate for modeling short flows is the model from [PFTK98], because of it pays far closer attention to retransmission timeouts than [LM97] and [MSMO97] and, unlike [Kum98], it assumes large RTTs. The approximate version of this model for steady-state TCP bandwidth, expressed in bytes per second, is:

$$B(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right) \tag{3}$$

where W_{max} is the maximum window allowed by receiver and sender (typically 8KB, 16KB, or 32KB), and the remaining parameters are as above.

To adapt this model to the case of short flows, the most obvious approach is to add the cost of the three-way handshake and delayed ACK, and then use this bandwidth estimate as the estimate for the bandwidth achieved for this flow once it starts sending data. This yields:

$$t = RTT + t_{delack} + data/B(p) \tag{4}$$

In the context of short flows, we will consider equation (4) to be the transfer time predicted by the [PFTK98] model.

4.2 A New Model for Short TCP Flows with Random Loss

The new model for short TCP flows that experience random loss is based on a few observations from simulations and the analytic results above:

- For short flows, the large initial value of the RTO timer will cause significant delays for flows that lose packets during connection setup.
- Because short flows start out in slow start, and even after losses, usually don't have time to grow to large $cwnd$ values, they will spend a majority of their time in slow start. This is in contrast to longer flows, which [MSMO97, PFTK98] showed are well-modeled as spending most of their time in congestion avoidance.
- Short flows will typically never have $cwnd$s big enough to allow them to use fast retransmit and fast recovery, so that they are well-modeled as a series of slow start periods followed by one or more retransmission timeouts. This is in contrast to longer flows, which will often recover from losses using fast retransmit [MSMO97, PFTK98].
- The number of losses suffered by a short flow suffering from i.i.d. random losses is described well by the binomial distribution (see figures 14 and 15 and the discussion in Section 5.3.1).
- Once you know the number of losses suffered by a flow, there is a smooth correlation between the number of losses and the transfer time. (see figure 16 and the discussion in section 5.3.2).

4.2.1 Data Transfer Model

We can use these observations to derive a model of transfer time, t, for a short flow transferring d bytes or $data = d/MSS$ packets over a path with an i.i.d. loss rate, p, using a TCP implementation with initial slow start window w and slow start exponential growth rate r, in the following manner: we model the data transfer as an initial connection establishment handshake, followed by alternating phases of slow start and successive retransmission timeouts (see figure 3).

First, note that a flow may experience any number, l, of losses, but we typically expect that the loss rate it sees – its effective loss rate $\frac{l}{data+l}$ – will be equal to the path loss rate, p. In this case we have:

$$\begin{aligned} \frac{l}{data+l} &= p \\ &\vdots \\ l &= \frac{data \cdot p}{1-p} \end{aligned} \tag{5}$$

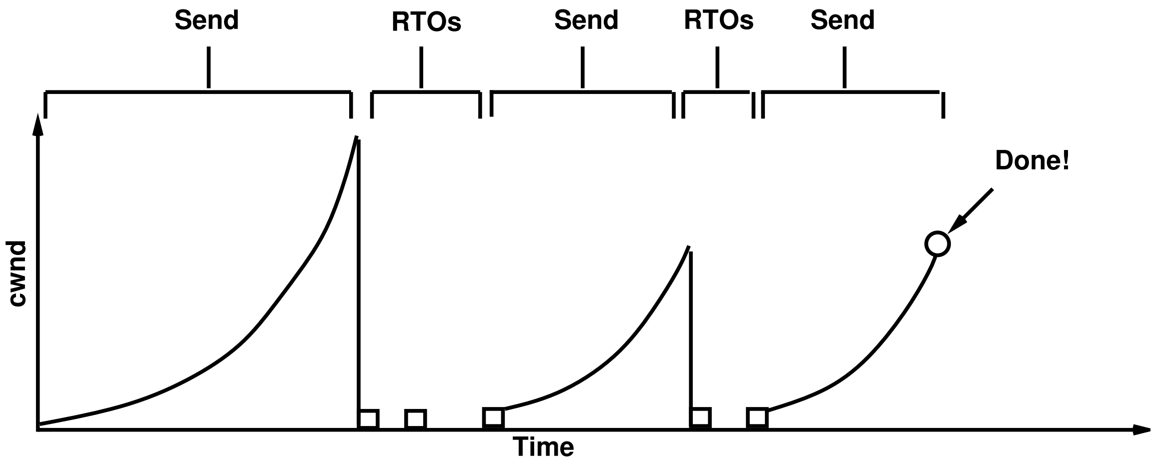

Figure 3: The phases of the short TCP flow model.

Next, we use the techniques from [PFTK98] to derive the probability, $Q(p)$, that a particular loss will incur a retransmission timeout. Using their result for congestion avoidance as an approximation to this quantity for short flows in slow start, we have:

$$Q(p) = \begin{cases} 0 & \text{if } p = 0 \\ \min\left(1, \frac{3}{\sqrt{\frac{8}{3bp}}}\right) & \text{if } p > 0 \end{cases} \tag{6}$$

The special case for $p = 0$ avoids a division by zero and ensures that the resulting model is defined for $p = 0$.

Using these two quantities we can approximate the number of retransmission timeouts experienced by a flow as:

$$n = l \cdot Q(p) \tag{7}$$

Now we need to know the number of runs of consecutive RTOs. The length of a run of consecutive RTOs has a geometric distribution with a mean of $\frac{1}{1-p}$. Thus the number, u, of runs of RTOs is:

$$\begin{aligned} u &= \frac{l \cdot Q(p)}{\frac{1}{1-p}} \\ u &= l \cdot Q(p) \cdot (1-p) \end{aligned} \tag{8}$$

From [PFTK98], the expected time for a single RTO run is:

$$t_u = T_0 \cdot \frac{1 + p + 2 * p^2 + 4 * p^3 + 8 * p^4 + 16 * p^5 + 32 * p^6}{1-p} \tag{9}$$

Thus the total time spent on RTOs is:

$$t_{RTO} = u \cdot t_u \tag{10}$$

If we consider the progress of a TCP connection as a series of phases where we are sending data, and each sending phase is separated from the next sending phase by one or more RTOs, then the number of phases in which we are actually sending useful data is:

$$v = u + 1 \tag{11}$$

And the average data sent per phase is:

$$e = \frac{data + l}{v} \tag{12}$$

Thus the time spent actually transferring data is:

$$t_{xfer} = v \cdot \log_r \left(\frac{e(r-1)}{w} + 1 \right) \cdot RTT \tag{13}$$

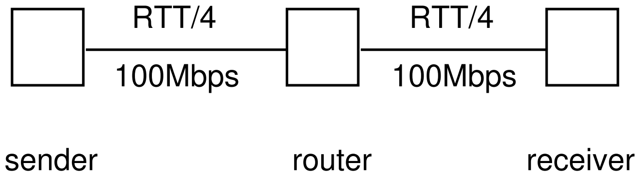

Figure 4: The topology used for ns simulations. The links are bi-directional. Each link has a one-way latency of $RTT/4$, so that the entire sender-receiver-sender path has the correct RTT value. The router is configured to produce i.i.d. random packet losses in each direction.

` Finally, the total transfer time, including handshake, compulsory delayed ACK, and time spent in timeout and actually sending data is:

$$t = RTT + t_{delack} + t_{RTO} + t_{xfer} \tag{14}$$

Note that this model has the nice property that it is defined for $p = 0$, whereas (4), the full version of $B(p)$ in [PFTK98], and the bandwidth model in [MSMO97] are all undefined for $p = 0$. In particular, if $p = 0$ then $Q(p) = 0$ and $u = 0$, meaning there are no RTOs or RTO runs; thus there are $v = 1$ phases where we send data. Hence the model in (14) reduces neatly and exactly to the lossless case (1) when $p = 0$.

From here on this model in (14) will be referred to as "the short flow model." Since the derivation above was simply aiming at a "typical" transfer time given the input parameters, it is not clear whether this model yields an expected time, or a median time, etc. Section 5 presents graphs showing how this model turns out to closely fit the median simulated transfer time across a range of values of transfer size, MSS, RTT, and loss rate. This section also compares this model with the ideal (1) and adapted [PFTK98] model (4).

4.2.2 Connection Establishment

In the short flow model (14), the time for the connection establishment handshake is given as one RTT. It is not always the case that handshakes will take a single RTT. In fact, if one of the SYN packets is lost, the sender must timeout and retransmit it. Since TCP has no RTT estimate yet, it has no idea how long to wait for an ACK of its SYN; consequently, the TCP requirement specifies that this initial timeout should be a conservative 3 seconds [Bra89]. As a result, a packet loss during connection establishment is extremely costly – the flow will suffer a 3 second penalty during a transfer that would probably otherwise take a few RTT (i.e., a fraction of a second).

It is important to be conscious of the extreme penalty for packet loss during connection establishment. On the other hand, such lengthy connection establishment costs are not typical. Since the model is trying to capture, in some sense, the typical behavior of a short connection, we neglect it for now.

5 Simulation Results

5.1 Methodology

In order compare these models with TCP implementations across the parameter space of transfer size, MSS, RTT, and loss rate in a controlled and repeatable fashion, we used the ns network simulator [ns]. We used the FullTCP model provided by ns. This model is based closely on the BSD TCP implementation, with a few variations involving round trip time estimation and retransmission timeouts. We modified this model to make its delayed acknowledgment more closely reflect actual delayed ACK implementations, and added code to record the motivation for sending each packet – retransmission timeout, triple-duplicate ACK, or "normal."

Figure 4 depicts the topology our simulations used. One sender sent packets to one receiver over a dedicated path containing one router. There was no competing cross-traffic. Instead, we configured the router to create random packet losses with an i.i.d. loss rate p_f in the forward (data) direction and p_b in the backward (ACK) direction; by default $p_f = p_b$.

Each experiment consisted of 4000 trials with different seeds for the random packet loss generation process; we settled on this number because using 10,000 trials did not seem to give different results, and each trial took

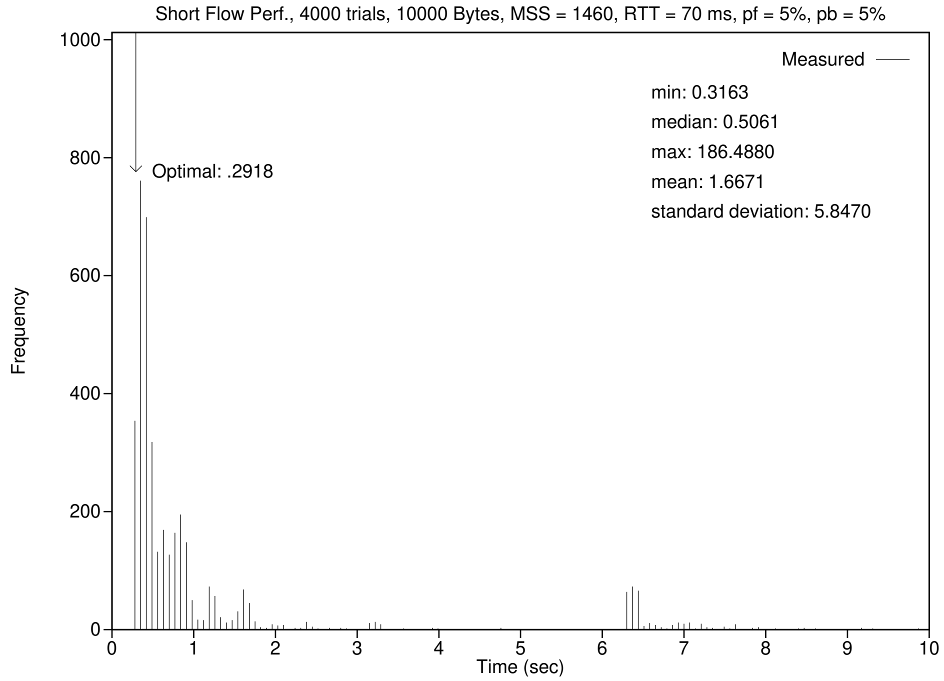

Figure 5: A histogram of short flow performance, for 10,000 bytes, MSS = 1460, p_f = 5%, p_b = 5%. Transferring only 7 packets with a low loss rate leads to a small variance in performance. Note the cluster of trials slightly after 6 seconds; nearly all of these trials encountered a packet loss during connection setup, resulting in a six-second timeout.

approximately a second of real time. In each trial a FullTCP agent on the sender opened a connection and immediately began sending the required amount of data. The experiment ended when the receiver received the last data segment.

We concentrated on the metric of data transfer time, by which we mean the time from when the sender opened the connection until the time when the receiver received the last data packet.

The default parameters for simulations were based on values that evidence suggests are representative of today's typical HTTP connections spanning the continental US: 10,000 bytes of data, MSS = 1460 bytes, 70ms RTT, and 5% loss rate in both directions [Ste96, TMW97, Pax97, BPS$^+$98]. Except where otherwise noted, these are the parameters used for each simulation.

5.2 Results

The most dramatic feature of the results is that the transfer times display heavy tails and high variance, particularly for long transfers with high loss rates. Figures 5 and 6 show histograms of flow performance for a 10,000-byte transfer. In Figure 5 the loss rate is moderate – 5% in each direction – and the MSS is 1460 bytes, so only 7 TCP segments are sent. Even here, the mean is three times the median, and the standard deviation is more than ten times the median time. In Figure 6 the loss rate is high – 10% in each direction – and the MSS is 536 bytes, forcing TCP to send 19 segments. Here the histogram is even more spread out – the median, mean, and standard deviation have each increased by approximately a factor of four from the previous scenario.

Most of this variation is due to the varied ways packet loss can affect the progress of a flow. Small variations in transfer time are caused by differences in when packet losses occur. Variations of about one RTT are caused when packet loss results in fast retransmission, and variations of small multiples of RTT are caused by retransmission timeouts. Flows that have one of their SYN segments lost suffer a 6-second timeout; flows that suffer such a six-second timeout and then *again* lose a SYN segment suffer a 12-second timeout on top of the first, due to TCP's exponential retransmission timer back-off. Other sources of delay include delayed acknowledgments and, in the real world, queuing delay.

Because of this high variance, the following graphs contain several reference points to convey the structure of the simulation results. Each graph contains the minimum transfer time, median transfer time, mean transfer time, and 80-th percentile transfer time. The 80-th percentile was chosen because for loss rates of around 5-10% the top 10-20-th

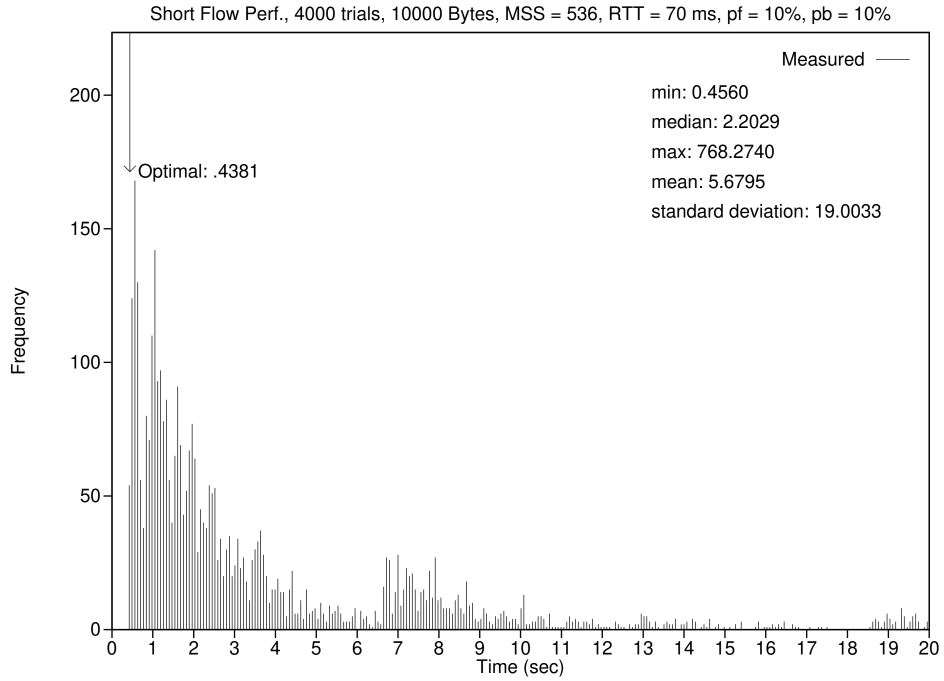

Figure 6: A histogram of short flow performance, for 10,000 bytes, $MSS = 536$, $p_f = 10\%$, $p_b = 10\%$. Transferring 19 packets with a high loss rate leads to highly varying performance. Note the clusters starting at 6.5 and 18.5 seconds. Most of the trials between 6.5 and 18 seconds suffered a six-second timeout during their first attempt at connection establishment. Trials beyond 18.5 seconds suffered a six-second and subsequent 12-second timeout.

percentile largely consist of flows that suffered a 6-second connection establishment timeout. In addition, each graph depicts the transfer time predicted by the optimal short flow model (1), the lossy short flow model (14), and the model from [PFTK98].

5.2.1 Transfer Size

Figure 7 shows the simulated and modeled performance of short TCP flows transferring between 500 and 32,000 bytes. The transfer time is basically logarithmic in the size of the data, because of the exponential growth of $cwnd$ during slow start. Note that the average and 80-th percentile are very large, due to the tail beyond six seconds. The optimal model fits the minimum transfer time well. The short flow model fits the median closely, and the [PFTK98] model fits almost as well, which is somewhat surprising.

Figure 8 shows the simulated and modeled performance of somewhat longer TCP flows transferring between 32,000 and 512,000 bytes. Note that the slope is now basically linear, indicating that the exponential growth of slow start is no longer the dominating factor. Here the flows are nearing steady-state, where transfer time is well-modeled as $t \approx data/bw$, for some steady-state bandwidth estimate, bw. This is to be expected, since with a 5% loss rate we should expect a flow to send 20 segments, or about 30KB, before losing a packet, timing out, and exiting slow start. With 5% losses in the reverse direction as well, this will happen even earlier, on average. Figure 8 also shows that the optimal model fits the minimum transfer time, until at 512KB all connections experience at least one loss. For longer flows, the short flow and steady-state model both gradually diverge from the median simulated performance; surprisingly, for these long flows the short flow model is closer to the median and average.

5.2.2 MSS

Figure 9 shows the simulated and modeled results for a reasonable range of MSS values [TMW97, Ste96]. Because all of TCP's $cwnd$ dynamics are in terms of MSS, changing the MSS used for a given data transfer simply has the effect of changing the number of packets the TCP implementation must send. Increasing the MSS by a factor of k is

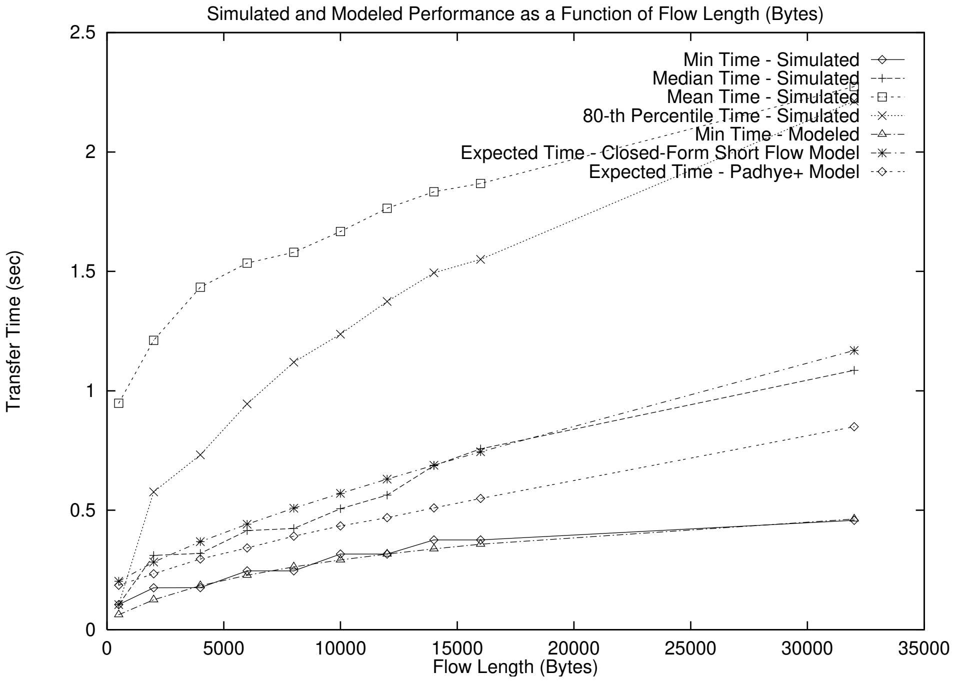

Figure 7: Varying transfer size from 500 bytes to 32,000 bytes. Note the logarithmic shape due to the exponential growth of slow start.

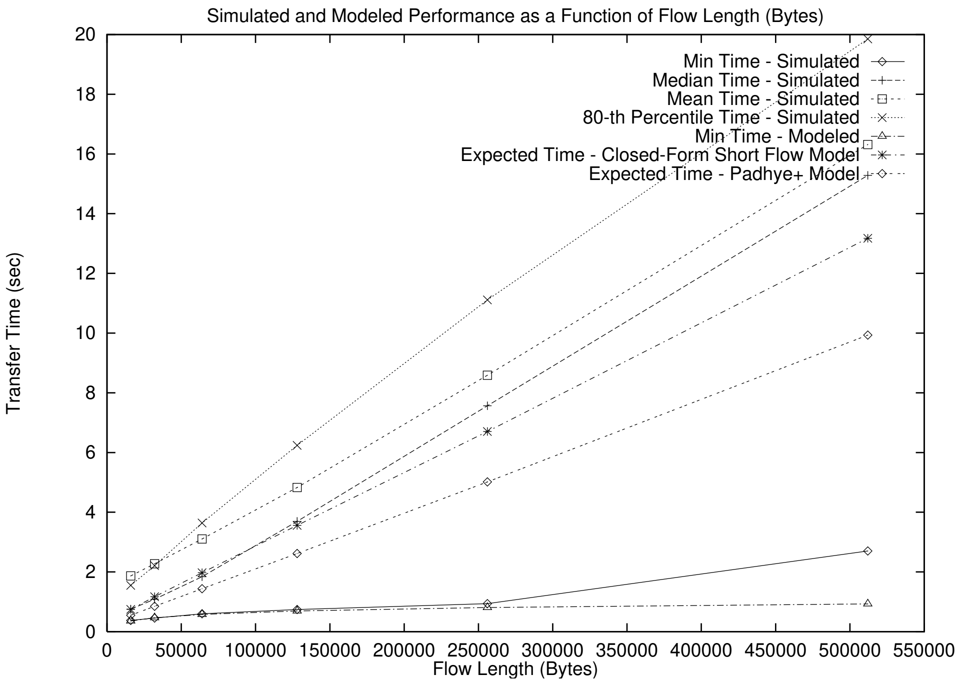

Figure 8: Varying transfer size from 32,000 bytes to 512,000 bytes. Note the linear shape, reflecting flows in steady-state. That is, transfer time is well-modeled as $t \approx data/bw$, for some steady-state bandwidth estimate, bw.

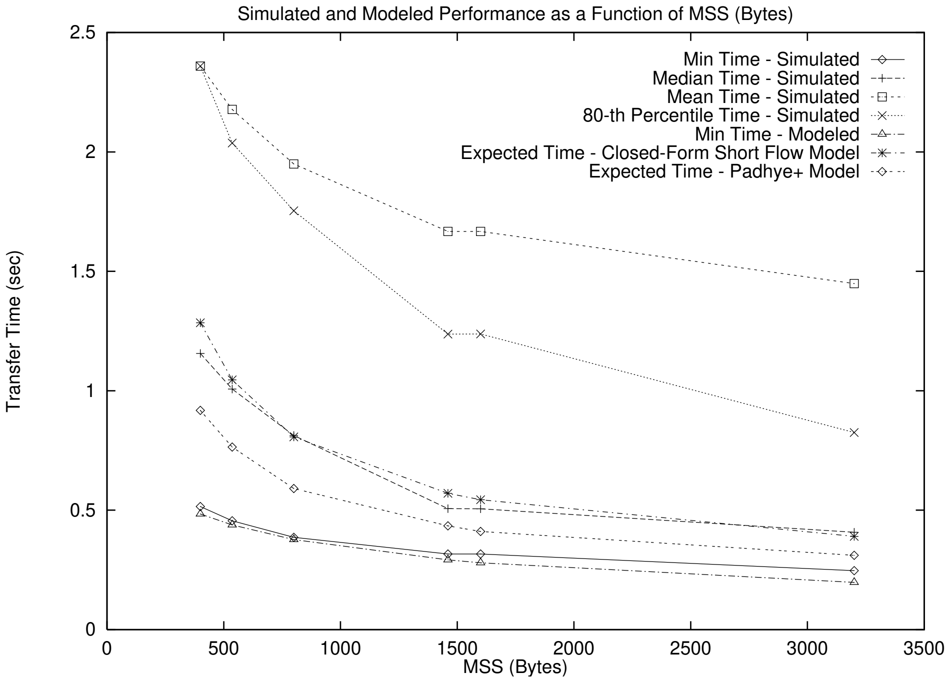

Figure 9: Varying MSS from 400 bytes to 3200 bytes.

roughly equivalent in performance to decreasing the size of the document by a factor of k. Thus figure 9 is roughly the inverse of figure 7.

5.2.3 RTT

Figure 10 shows shows the simulated and modeled results for a reasonable range of RTT values [Ste96]. Because nearly all of TCP's $cwnd$ dynamics happen on RTT time scales – the exceptions being delayed ACKs and retransmission timeouts – transfer time is essentially linear in RTT. All the models fit the simulated results fairly well, except the the short flow model diverges as RTT grows to 200ms and 400ms. This is probably because for large RTT values the variance of retransmission timeouts increases dramatically in ways not captured by the model.

5.2.4 Loss Rate

Figures 11, 12, and 13 show performance as a function of loss rate. In each case, $p_f = p_b$. Further simulations showed that small variations in the loss rate in the reverse (ACK) direction made little difference. This is to be expected, since TCP ACKs are cumulative, meaning that the information in a lost ACK is carried implicitly by the next ACK.

Figure 11 shows performance for 10KB transfer with loss rates of 1-15%. All the models fit the median fairly well, even when, at high loss rates, the mean skyrockets because of repeated retransmission timeouts.

Figures 12 and 13 depict the performance of 10,000-byte and 128,000-byte flows, respectively, under conditions of low loss rate. Notice how, in each case, the short flow model is better at capturing the performance of flows under very low loss rates. This is the most marked difference between the predictions of the two models, because at very low loss rates a steady-state model predicts very high performance, whereas a short flow model captures the fact that a short flow must spend several RTTs ramping up to speed. This said, the steady-state model is not too far off; see Section 5.4 for a discussion.

5.3 Simulation Results and the Basis of the Short Flow Model

Section 4 described how the short flow model starts from an estimate of the number of packet losses experienced by a connection and derives the transfer time by modeling the transfer time for a given amount of data given a certain number of losses. Section 5.3.1 describes the motivation behind the estimation of the number of packet losses, and

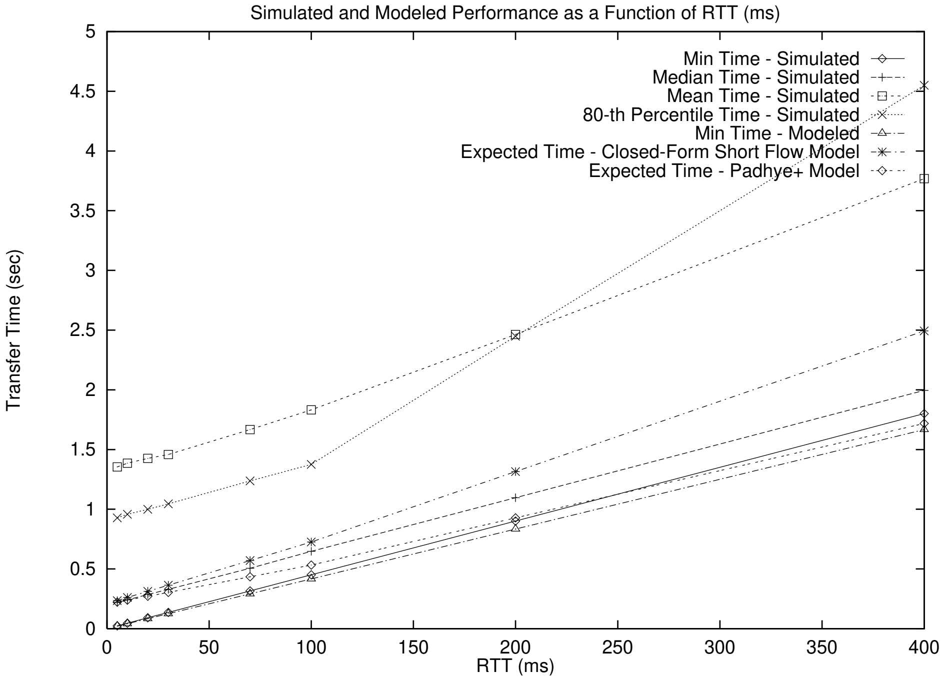

Figure 10: Varying RTT from 5ms to 400ms.

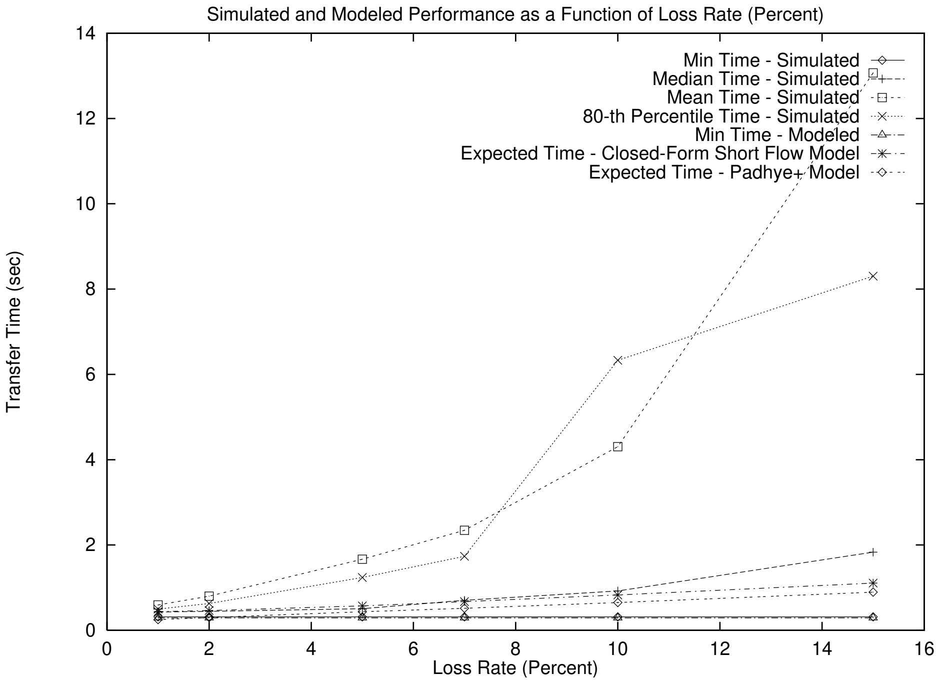

Figure 11: Varying loss rate from 1% to 15%

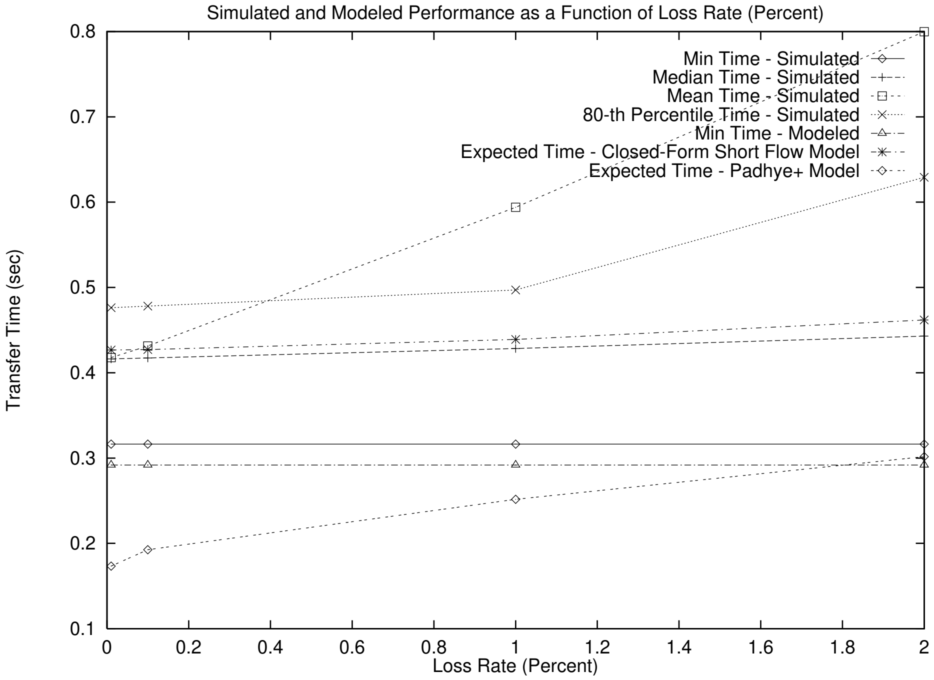

Figure 12: Varying loss rate from 0.01% to 2%, for 10,000-byte transfers.

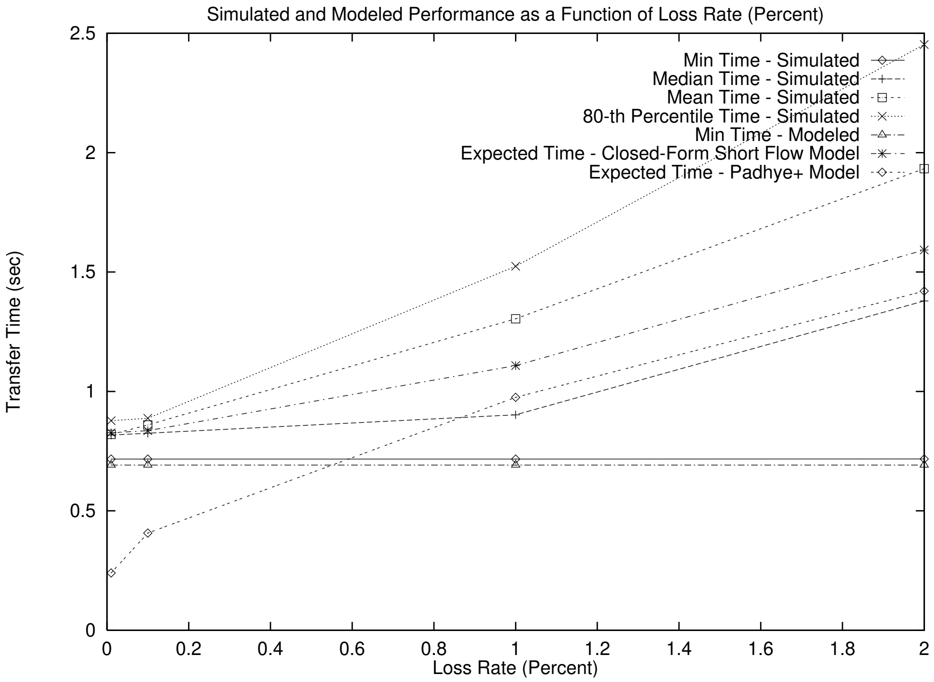

Figure 13: Varying loss rate from 0.01% to 2%, for 128,000-byte transfers.

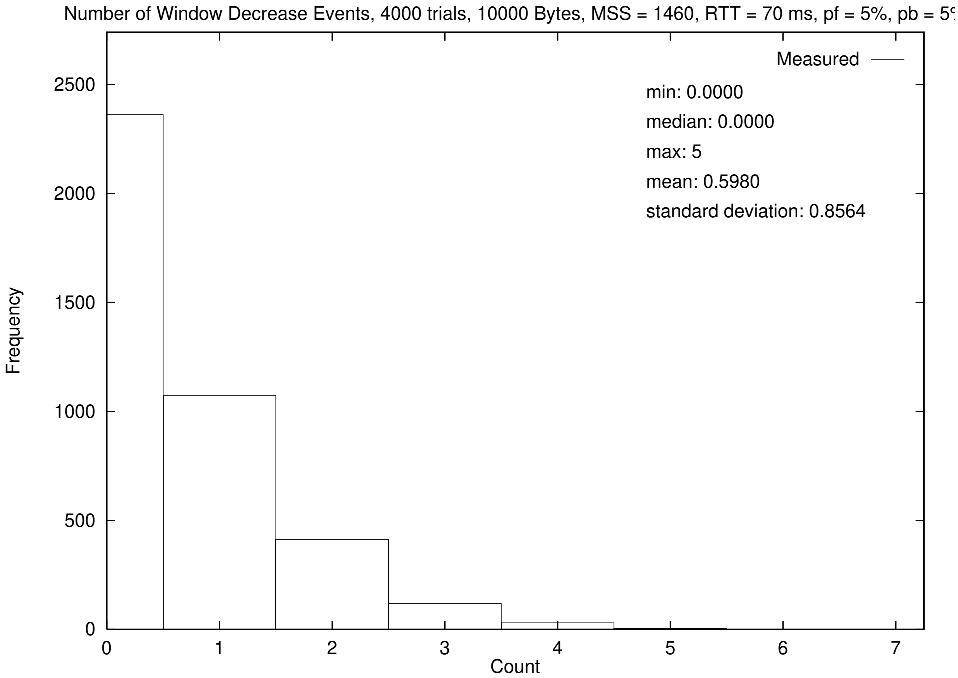

Figure 14: Binomial distribution of loss-induced window decrease events – retransmission timeouts or triple-duplicate ACK events – for a 10,000-byte flow.

Section 5.3.2 describes the motivation behind the estimation of the transfer time based on the number of packet losses. Together these two sections describe the key observations that lead to the short flow model (14).

5.3.1 Number of Losses

Figures 14 and 15 show the distribution of the number of loss-induced window decrease events – retransmission timeouts or triple-duplicate ACK events – for 10,000-byte and 128,000-byte flows respectively. Because the simulations use i.i.d. packet losses, the number of window decrease events is modeled very well as a binomial distribution. As a result, short flows often experience no packet losses, so that they are well-modeled by the lossless model; longer flows usually experience a few losses. This is reflected born out by figure 8. In either case – short flow or long – the critical aspect for the model is that the distribution of the number of losses is well-characterized by the average number of losses. This is true for the loss model in the simulation, but is likely not true for the bursty loss patterns of the Internet [Pax97].

5.3.2 Performance as a Function of Number of Packet Losses

Figure 16 is a scatter plot showing how the number of window decrease events (excluding losses during connection setup) correlates with the transfer time required for 64,000 bytes. To clarify the trend we plot the median and mean time for each column, excluding trials that suffered packet losses during connection setup, since these have disproportionate effects and are easily characterized by other means. The plot shows that transfer time appears to increase polynomially as a function of the number of losses. The reason is that, as the number of losses increases, there are more and more consecutive losses, leading to exponential back-off and a higher cost, on average, for each timeout.

5.4 Simulation Results and the Steady-State Model

According to the simulation results, equation (4), the short-flow adoption of the steady-state TCP performance model from [PFTK98], predicts the performance of short TCP flows surprisingly well, given that the model is based on the assumption that the flow is very long. How can this be?

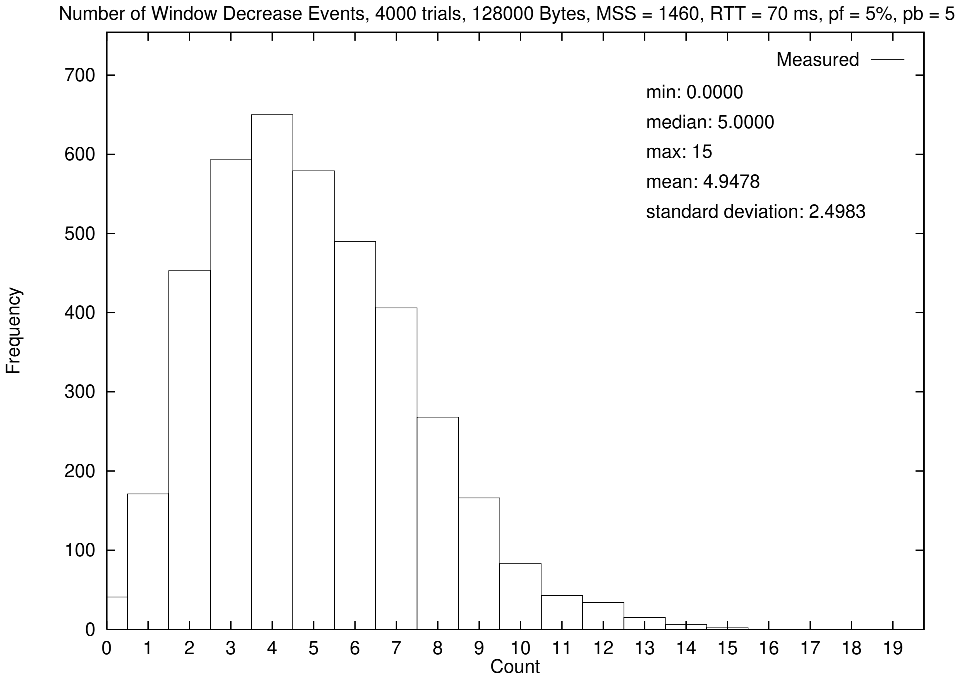

Figure 15: Binomial distribution of loss-induced window decrease events – retransmission timeouts or triple-duplicate ACK events – for a 128,000-byte flow.

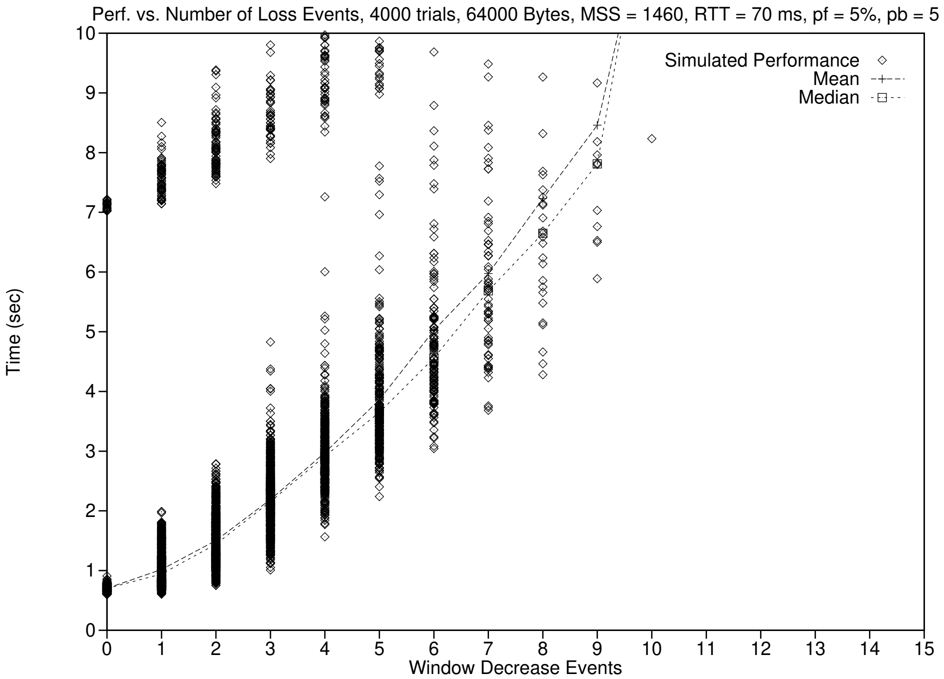

Figure 16: Transfer time as a function of the number of loss events. Each loss is more costly than the last, due to the increasing frequency of consecutive loss and the resulting exponential back-off.

remote host	location	ping RTT	sustained BW
`www.cs.princeton.edu`	Princeton, NJ	108 ms	200 KB/s
`www.ucdavis.edu`	Davis, CA	50 ms	470 KB/s
`www.utexas.edu`	Austin, TX	62 ms	350 KB/s
`www.vt.edu`	Blacksburg, VA	109 ms	200 KB/s

Table 1: The destination hosts use for short flow transfers.

For short flows experiencing moderate levels of loss (1-15%), it is not surprising that the models agree, for several reasons. First, since connections that experience loss early will reach something approaching steady state early on. Second, retransmission timeouts are a significant fraction of transfer time, and the short flow model borrows its model of retransmission timeouts from the steady-state model in [PFTK98]. Third, for high loss rates, the period between timeouts is short and $cwnd$ is small, so it doesn't much matter whether you model this period as consisting of congestion avoidance (as in the steady-state model) or slow start (as in the short flow model).

For short flows experiencing packet loss rates below 1%, the steady-state model does not fit as well (see again figures 12 and 13) but it is still only a factor of two or three off. How can it be so close?

First, suppose we include in the steady-state model, as we do in 4, the connection-establishment handshake. Because the three-way-handshake takes one RTT, and short transfers typically only take a few RTT, simply by including the handshake overhead, a steady-state model is well on its way to predicting a transfer time that is within 1/5 to 1/8 of the actual time for a short flow. This is what is happening in figures 12 and 13 for loss rates of 0.1%.

Another factor that limits the bandwidth prediction of a steady-state model enough to keep its transfer time prediction bounded close to the actual value is window limitations. TCP will never send more than $maxwin$ bytes per RTT, where $maxwin$ is some maximum window size determined by the size of the sending and receiving socket buffers. Typically $maxwin$ is 4KB, 8KB, 16KB, or 32KB, but very rarely more than 64KB, since only 16 bits are available in the TCP header to indicate the receive window, without using special options [JBB92, Ste96]. Consider a case with a low loss rate where $maxwin$ is 32KB and we are using a steady-state model to predict the transfer time for 128KB. In this case the steady-state model will predict that the data can be transferred in no fewer than 4 RTT (32KB per RTT), whereas the short flow model will predict something like 10 RTTs. Again, serendipitously, the steady-state model is only off by a small factor. This is happening in figures 12 and 13 for loss rates of 0.01%.

6 Measurement Results

In order to compare the models against real world TCP implementations and Internet paths, we conducted several measurements. Each trial consisted of a short TCP transfer of a random amount of data, selected uniformly between 1 and 128,000 bytes, from a PC in Seattle at the University of Washington running Linux 2.0.35 to a machine at another site on the Internet. Table 6 lists the four destination sites measured, along with a sample of the RTT for a 1400-byte ping to that site and a sample of the TCP bandwidth from UW to that site using a 1 MB transfer. These sites were chosen for two reasons: they were running a `discard` service that allowed us to perform TCP transfers of arbitrary sizes, and they all had large enough receive windows that the receive window did not limit throughput.

During each trial we used `tcpdump` to capture a trace of the TCP segments sent in order to measure the transfer time, MSS, RTT, RTO, and average loss rate. We modified the Linux kernel to record microsecond-resolution time stamps on incoming and outgoing packets, instead of the default 10ms-granularity time stamps.

Each 7.5-hour experiment consisted of a sequence of transfer trials, where the random inter-trial times were exponentially distributed with a mean of 60 seconds. This yielded about 300-400 trials for each site. The experiments were run between 9:30AM and 5PM PDT on a weekday.

Figures 17, 18, 19 and 20 show the results of the experiments, comparing the transfer times of the trials against three models: the lossless short flow model (1), the adapted steady-state model (4), and the short flow model (14). The lossless ideal model (1) and short flow model (14) with $w = 2$ and $r = 1.5$ both provide reasonably close fits. The fit for the adapted steady-state model (4) is poor in each case. This is because the low loss rates cause this model to predict a fairly high steady-state bandwidth, which translates into a short transfer time; as expected, this model does not reflect the dominant effect of slow starting from a very small window.

In figure 17 the fit is not very tight. This appears to be because this host's TCP implementation has an adaptive delayed ACK policy. It ACKs every packet for the first few packets, which results in slightly faster growth of $cwnd$

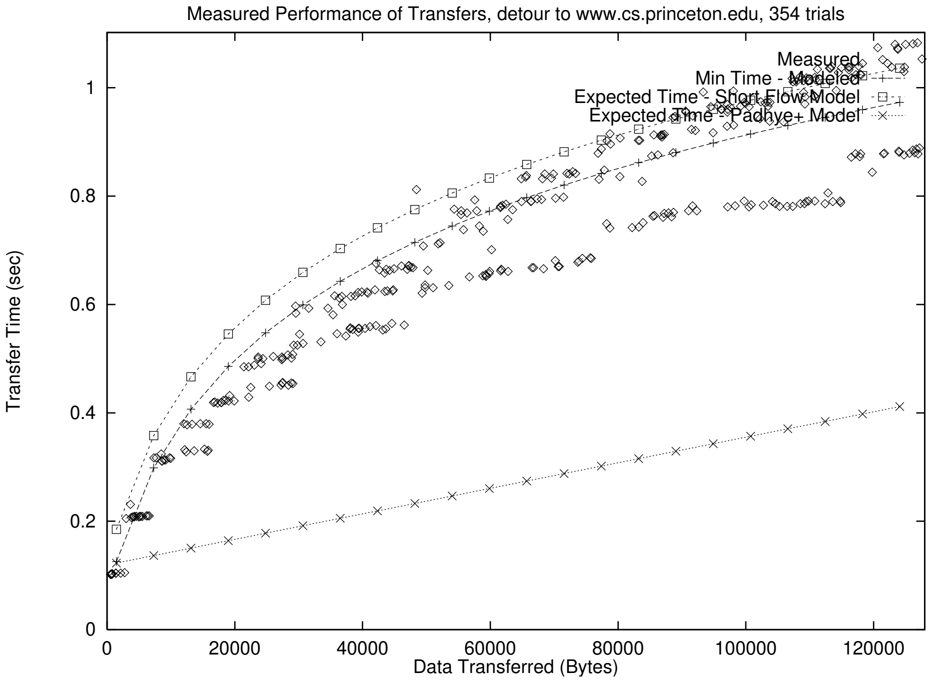

Figure 17: Results for 354 transfers to `www.cs.princeton.edu`. MSS = 1460 bytes, average RTT = 119 ms, average RTO = 206 ms, average loss rate = 0.06%, average receiver window = 64KB. The fit is not very tight. This appears to be because this host's TCP implementation has an adaptive delayed ACK policy. It ACKs every packet for the first few packets, which results in slightly faster growth of $cwnd$ and a slightly lower transfer time.

and a slightly lower transfer time.

7 Limitations

There are several limitations with the models proposed in sections 3 and 4:

- Both models assume a high-bandwidth, high-delay path. While this is typical of paths with T1 or better bottleneck bandwidth, it is not typical of modems, for example. TCP performance over modems is more dominated by transmission time, though the large and varying RTT may inflate RTOs as well.
- The lossy model (14) assumes that losses are well-modeled as independent. [Pax97] suggests that losses are actually not well-modeled as independent; multiple concurrent connections started by web browsers will have even more correlations in their losses.
- Neither model incorporates queuing delay. For low bandwidth paths – paths with a T1 or lower bandwidth – queuing delay will be significant, especially in the way it inflates the RTT variance, and thus the duration of RTOs. For high bandwidth paths, this will be much less significant. For example, a queue of 50 1500-byte packets can be drained in 4ms by a 155Mbps link.
- The models, simulations, and measurements only considered the Reno version of TCP. Reno is the most popular version today, but other versions, such as Tahoe, New Reno, Vegas, SACK, and FACK, may have slightly different performance [FF96, MM96].

8 Conclusion

This paper compared several different models for the performance of short TCP flows under realistic loss conditions: an adaptation of an existing steady-state model (4), a lossless short flow model (1), and a new model for short flows

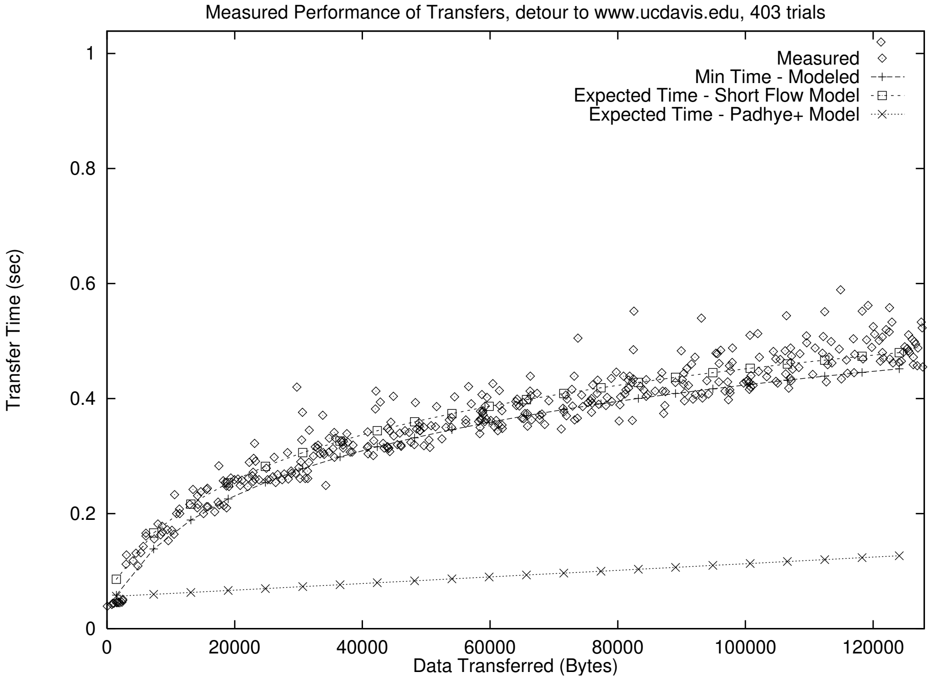

Figure 18: Results for 403 transfers to `www.ucdavis.edu`. MSS = 1460 bytes, average RTT = 55 ms, average RTO = 203 ms, average loss rate = 0.02%, average receiver window = 32KB.

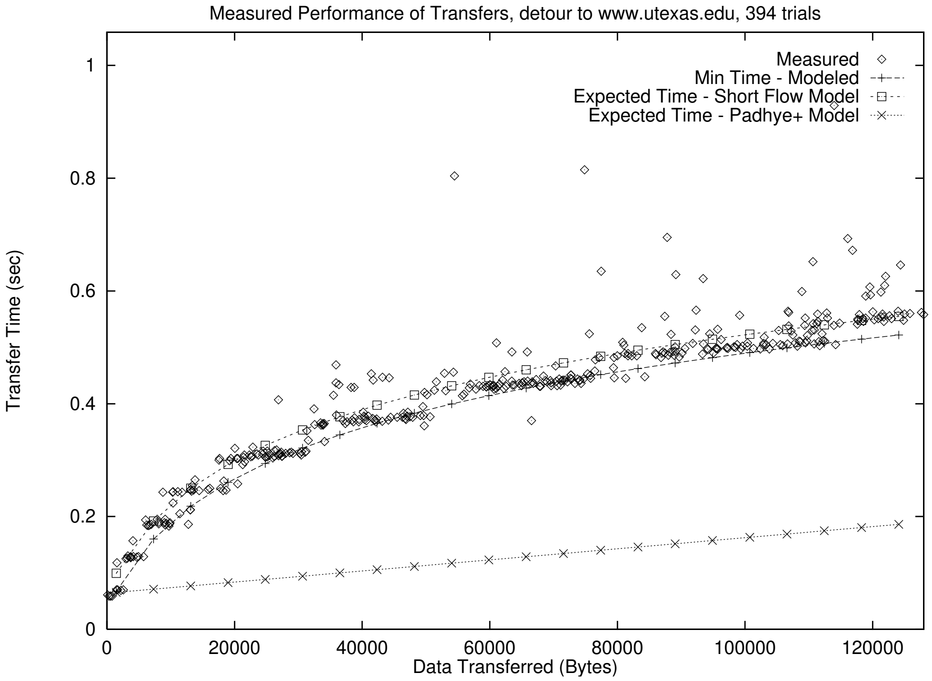

Figure 19: Results for 394 transfers to `www.utexas.edu`. MSS = 1460 bytes, average RTT = 64 ms, average RTO = 150 ms, average loss rate = 0.04%, average receiver window = 33KB.

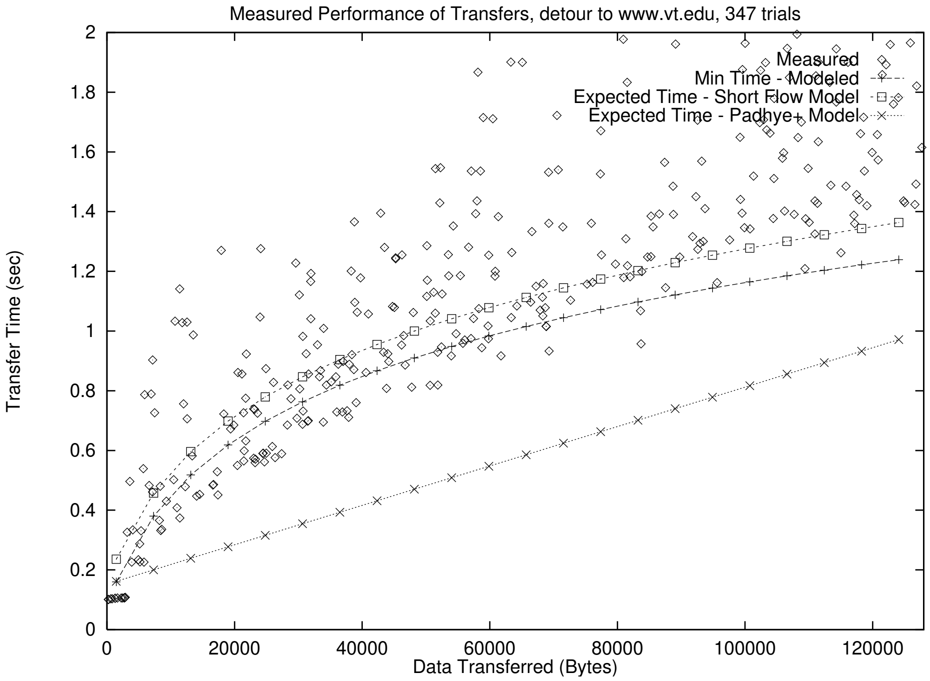

Figure 20: Results for 347 transfers to `www.vt.edu`. MSS = 1460 bytes, average RTT = 152 ms, average RTO = 390 ms, average loss rate = 0.3%, average receiver window = 30KB. The wide variance in transfer times is due to the loss rate.

that experience losses (14). We found that:

- The lossless short flow model (1) provides a very close fit to the simulated and measured performance of short flows that experience no loss, but provides a poor fit to the performance of flows that experience losses.
- The steady-state model (4) adapted from [PFTK98] provides a surprisingly close fit for simulated short flows that experience a moderate-to-high loss rate, but provides a poor fit to measured performance. Furthermore, this model is inconvenient for short flows because it is undefined for zero loss rates.
- By borrowing ideas from [PFTK98] and applying them to short flows, the new short flow model (14) fits short flows that experience 0-15% loss rate in the simulated and measured scenarios. When applied to lossless scenarios, this model reduces to the lossless short flow model.

Finally, the new short flow model is similar enough to the steady-state model from [PFTK98] that it may be possible to combine the two approaches, yielding a single model that predicts the performance of TCP flows of any length experiencing any level of loss.

References

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

[AHO98] Mark Allman, Chris Hayes, and Shawn Ostermann. An evaluation of TCP with larger initial windows. *ACM Computer Communications Review*, 28(3), July 1998.

[All98] Mark Allman. On the generation and use of TCP acknowledgments. *ACM Computer Communications Review*, 28(5), October 1998.

[BC94] Hans-Werner Braun and Kimberly C. Claffy. Web traffic characterization: An assessment of the impact of caching documents from ncsa's web server. In *Proceedings of the Second World Wide Web COnference '94*, October 1994.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. RFC 1945, May 1996.

[BOP94] Larry S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM '94*, London, UK, August 31st - September 2nd 1994.

[BPS+98] Hari Balakrishnan, Venkata Padmanabhan, Srinivasan Seshan, Randy H. Katz, and Mark Stemm. TCP behavior of a busy internet server: Analysis and improvements. In *Proceedings of IEEE INFOCOM '98*, April 1998.

[Bra89] R. Braden. Requirements for internet hosts – communication layers. RFC 1122, October 1989.

[BSSK97] Hari Balakrishnan, Mark Stemm, Srinivasan Seshan, and Randy H. Katz. Analyzing stability in wide-area network performance. In *Proceedings of the 1997 SIGMETRICS Conference*, June15–18 1997.

[CBC95] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical Report BU-CS-95-010, Boston University, July 1995.

[FF96] K. Fall and S. Floyd. Simulation-based comparisons of tahoe, reno, and SACK TCP. *ACM Computer Communications Review*, 26(3), July 1996.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2068, January 1997.

[Flo91] Sally Floyd. Connections with multiple congested gateways in packet-switched networks, part 1: One-way traffic. *ACM Computer Communications Review*, 21(5), October 1991.

[GB97] Steven D. Gribble and Eric A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, December 1997.

[GCMW98] Rajarshi Gupta, Mike Chen, Steven McCanne, and Jean Walrand. WebTP: A receiver-driven web transport protocol. Submitted to INFOCOM 99, July 1998.

[Hei97] John Heidemann. Performance interactions between P-HTTP and TCP implementations. *ACM Computer Communications Review*, 27(2), April 1997.

[Hoe96] Janey C. Hoe. Improving the start-up behavior of a congestion control sheme for TCP. In *Proceedings of ACM SIGCOMM '96*, pages 270–280, August26–30 1996.

[HOT97] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking*, 5(5), October 1997.

[HSMK98] Thomas Henderson, Emile Sahouria, Steven McCanne, and Randy Katz. On improving the fairness of TCP congestion avoidance. In *Proceedings of IEEE Globecom `98*, November 1998.

[Jac88] Van Jacobson. Congestion avoidance and control. *Proceedings of ACM SIGCOMM '88*, 18, 4, August 1988.

[JBB92] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

[jum98] Jumbo frames information. `http://www.alteon.com/products/jumbo_frames.html`, October 1998.

[Kum98] Anurag Kumar. Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[LM97] T.V. Lakshman and Upamanyu Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, June 1997.

[Mah97a] Bruce A. Mah. An empirical model of HTTP network traffic. In *Proceedings of IEEE INFOCOM '97*, April 1997.

[Mah97b] Jamshid Mahdavi. TCP performance tuning. `http://www.psc.edu/networking/tcptune/slides/`, April 1997.

[MM96] M. Mathis and J. Malidavi. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of ACM SIGCOMM '96*, August26–30 1996.

[Mog95] Jeff C. Mogul. The case for persistent-connection HTTP. *ACM Computer Communications Review*, 25(4), October 1995.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm). *ACM Computer Communications Review*, 27(3):67–82, July 1997.

[NGBS+97] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of ACM SIGCOMM '97*, September 1997.

[ns] UCB/LBNL/VINT network simulator - ns (version 2).

[OKM96] Teunis Ott, J.H.B. Kemperman, and Matt Mathis. The stationary behavior of ideal TCP congestion avoidance. `ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps`, August 1996. Unpublished manuscript.

[Pax97] Vern Paxson. End-to-end Internet packet dynamics. In *Proceedings of ACM SIGCOMM '97*, September14–18 1997.

[PFTK98] 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*, pages 303–314, Vancouver, BC, September 1998.

[PK98] Venkata N. Padmanabhan and Randy H. Katz. TCP fast start: A technique for speeding up web transfers. In *Proceedings of the IEEE Globecom '98 Internet Mini-Conference*, November 1998.

[PN98] Kedarnath Poduri and Kathleen Nichols. Simulation studies of increased initial TCP window size. RFC 2415, September 1998.

[SAA+98] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: a case for informed internet routing and transport. Technical Report UW-CSE-98-10-05, University of Washington, Seattle, October 1998.

[SAP98] W. Stevens, M. Allman, and V. Paxson. TCP congestion control. `http://www.ietf.org/internet-drafts/draft-ietf-tcpimpl-cong-control-00%.txt`, August 1998.

[SP98] T. Shepard and C. Partridge. When TCP starts up with four packets into only three buffers. RFC 2416, September 1998.

[SSK97] Srinivasan Seshan, Mark Stemm, and Randy H. Katz. SPAND: Shared Passive Network Performance Discovery. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, 1997.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison Wesley, 1994.

[Ste95] W. Richard Stevens. *TCP/IP Illustrated*, volume 2. Addison Wesley, 1995.

[Ste96] W. Richard Stevens. *TCP/IP Illustrated*, volume 3. Addison Wesley, 1996.

[Ste97] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, January 1997.

[TMW97] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November 1997.

[Tou97] Joe Touch. TCP control block interdependence. RFC 2140, April 1997.