Modeling TCP Latency

Neal Cardwell, Stefan Savage, Thomas Anderson
{cardwell, savage, tom}@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195 USA

Abstract—**Several analytic models describe the steady-state throughput of bulk transfer TCP flows as a function of round trip time and packet loss rate. These models describe flows based on the assumption that they are long enough to sustain many packet losses. However, most TCP transfers across today's Internet are short enough to see few, if any, losses and consequently their performance is dominated by startup effects such as connection establishment and slow start. This paper extends the steady-state model proposed in [34] in order to capture these startup effects. The extended model characterizes the expected value and distribution of TCP connection establishment and data transfer latency as a function of transfer size, round trip time, and packet loss rate. Using simulations, controlled measurements of TCP transfers, and live Web measurements we show that, unlike earlier steady-state models for TCP performance, our extended model describes connection establishment and data transfer latency under a range of packet loss conditions, including no loss.**

I. Introduction

Many of today's popular Internet applications, including the World-Wide Web, e-mail, file transfer, Usenet news, and remote login, use TCP as a transport protocol. As a consequence, TCP controls a large fraction of flows, packets, and bytes that travel over wide-area Internet paths [41, 10].

Recently researchers have proposed a number of analytic models to characterize TCP performance in terms of round-trip delay and packet loss rate [12, 24, 18, 27, 22, 25, 21, 34, 30, 33, 40, 9]. Beyond achieving a better understanding of the sensitivity of TCP performance to network parameters, these models have helped inform the design of active queue management schemes [13, 32] and TCP-friendly multicast protocols [6, 42].

The analytic models proposed to date can be split into two broad classes: models for steady-state bulk transfer throughput, and models for short flows that suffer no packet loss.

The majority of models fit in the first class; they focus on characterizing bulk transfer throughput. While these models are very successful at predicting steady-state throughput [5, 38], many recent studies have noted that the majority of TCP flows traveling over the wide-area Internet are very short, with mean sizes around 10KB and median sizes less than 10KB [11, 4, 23, 16, 41, 10]. Because these flows are so short, they often spend their entire lifetime in TCP's slow start mode, without suffering a single loss. Since the steady-state models assume flows suffer at least one loss, they are undefined for this common case.

The second class of models focuses on these short flows that suffer no packet losses [18, 24, 35]. However, these models do not consider delayed acknowledgments, sender or receiver buffering limitations, alternative initial congestion windows, or losses during connection establishment, each of which can have a dramatic performance impact.

This paper proposes a new model for TCP performance that integrates the results from both classes of models. In particular, we extend the steady-state results from [34] by deriving new models for two aspects that can dominate TCP latency: the connection establishment three-way handshake and slow start. Using simulation, controlled measurements, and live Web traces we show that our new slow start model works well for TCP flows of any length that suffer no packet losses, and the model from [34] often works well for flows of any length that do suffer packet losses. Thus our combined approach, which integrates these two models, is appropriate for predicting the performance of both short and long flows under varying packet loss rate conditions. In addition, we suggest a technique for estimating the distribution of of data transfer latencies.

The rest of this paper is organized as follows. Section II describes the new model and relates it to the models from which it is descended. Section III compares the connection establishment model with simulations and Section IV compares the data transfer model to simulations, TCP measurements, and HTTP traces. Finally, Section V summarizes our conclusions.

II. The Model

A. Assumptions

The extended model we develop here has exactly the same assumptions about the endpoints and network as the steady state model presented in [34]. The following section describes these assumptions in detail, including a few assumptions not stated explicitly in [34], since these details can have a large impact on the latency of short TCP flows. Throughout our presentation of this model we use the same terminology and notation as [34].

A.1 Assumptions about Endpoints

First, we assume that the sender is using a congestion control algorithm from the TCP Reno family; we refer readers to [37, 2, 20] for details about TCP and Reno-style congestion control. While we describe the model in terms of the simpler and more common TCP Reno algorithm, it should apply just as well to newer TCP implementations using NewReno, SACK, or FACK [14, 28, 26]. Previous measurements suggest the model should be even more faithful to these more sophisticated algorithms, as they are more resilient to bursts of packet losses [27, 5].

Since we are focusing on TCP performance rather than general client-server performance, we do not model sender or re-

0-7803-5880-5/00/$10.00 (c) 2000 IEEE

ceiver delays due to scheduling or buffering limitations. Instead, we assume that for the duration of the data transfer, the sender sends full-sized segments (packets) as fast as its congestion window allows, and the receiver advertises a consistent flow control window. Similarly, we do not account for effects from the Nagle algorithm or silly window syndrome avoidance, as these can be minimized by prudent engineering practice [17, 29].

We assume that the receiver has a "typical" delayed acknowledgment implementation, whereby it sends an acknowledgment (ACK) for every $b = 2$ data segments, or whenever its delayed ACK heartbeat timer expires, whichever happens first. Although Linux 2.2, for example, uses a more adaptive implementation of delayed ACKs, for very short flows this technique can be modeled well with $b = 1$, and for longer flows the effect of this approach is only to shave off at most one or two round trips.

A.2 Assumptions about the Network

We model TCP behavior in terms of "rounds," where a round starts when the sender begins the transmission of a window of packets and ends when the sender receives an acknowledgment for one or more of these packets. We assume that the time to send all the packets in a window is smaller than the duration of a round and that the duration of a round is independent of the window size. Note that with TCP Reno congestion control this can only be true when the flow is not fully utilizing the path bandwidth.

We assume that losses in one round are independent of the losses in any other round, while losses in the same round are correlated, in the sense that any time a packet is lost, all further packets in that round are also lost. These assumptions are idealizations of observations of the packet loss dynamics of paths using FIFO drop-tail queuing [7, 36, 44] and may not hold for links using RED queuing [15] or paths where packet loss is largely due to link errors rather than congestion. We assume that the probability of packet loss is independent of window size; again, this can only hold for flows that are not fully utilizing a link.

When modeling data transfer, we assume that packet loss happens only in the direction from sender to receiver. This assumption is acceptable because low levels of ACK loss have only a small effect with large windows, and network paths are often much more congested in the direction of data flow than the direction of ACK flow [41, 39]. Because packet loss has a far more drastic result during connection establishment, we model packet loss in both directions when considering connection establishment.

A.3 Assumptions about the Transfer

Though we share the assumptions of [34] about the endpoints and network, we relax several key assumptions about the data transfer. Namely, we allow for transfers short enough to suffer a few packet losses, or zero losses, and thus to be dominated by connection establishment delay and the initial slow start phase.

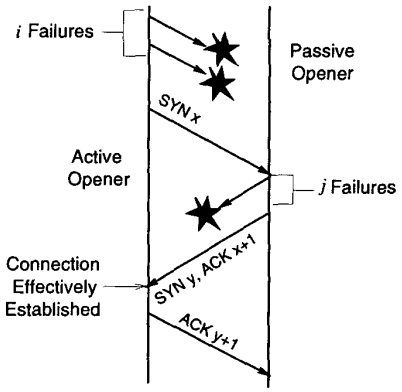

Fig. 1. TCP connection establishment example.

B. Model Overview

Our model describes two aspects of TCP performance. First, we derive expressions for the expected value and distribution of time required for the connection establishment handshake that begins a TCP connection. Second, we derive an expression for the expected latency to transfer a given amount of data and then describe our methodology for extrapolating the distribution of this transfer latency. Separating connection establishment latency from data transfer latency allows us to apply the model to applications that establish a single TCP connection and use it for several independent data transfers.

C. Connection Establishment

Every successful TCP connection begins with a "three-way handshake" in which the endpoints exchange initial sequence numbers. Figure 1 shows an example. The initiating host, typically the client, performs an *active open* by sending a SYN segment with its initial sequence number, x. The server performs a *passive open*; when it receives a SYN segment it replies with a SYN segment of its own, containing its initial sequence number, y, as well as an ACK for the active opener's initial sequence number. When the active opener receives this SYN/ACK packet, it knows that the connection has been successfully established. It confirms this by sending an ACK of the passive opener's initial sequence number. At each stage of this process, if either party does not receive the ACK that it is expecting within some SYN timeout, T_s, initially three seconds [8], it retransmits its SYN and waits twice as long for a response.

To model this process in the presence of packet loss in either direction, we define p_f as the "forward" packet loss rate along the path from passive opener to active opener ("forward" since this is usually the primary direction of data flow) and p_r as the "reverse" packet loss rate. Let RTT be the average round trip delay between the two hosts.

Our model of the three-way handshake consists of the following stages. First, the active opener transmits its SYN $i \geq 0$ times unsuccessfully, until the $(i+1)$-th SYN arrives successfully at the passive opener. Next the passive opener will ignore further

SYNs from the active opener while it repeatedly retransmits its SYN/ACK until it receives a response. In general it will send its SYN/ACK $j \geq 0$ times unsuccessfully until finally the $(j+1)$-th SYN/ACK arrives successfully at the active opener. For the purposes of the model, we consider the connection to be established at this point, since, in most application protocols, immediately after sending the ACK $y+1$, the active opener sends a data segment to the passive opener that contains a redundant ACK $y+1$.

Let $P_h(i,j)$ be the probability of having a three-way handshake episode consisting of exactly i failures transmitting SYNs, followed by one successful SYN, followed by exactly j failures transmitting SYN/ACKs, followed by one successful SYN/ACK. Then

$$P_h(i,j) = p_r^i \cdot (1-p_r) \cdot p_f^j \cdot (1-p_f) \tag{1}$$

The latency, $L_h(i,j)$, for this process is

$$\begin{aligned} L_h(i,j) &= RTT + \left(\sum_{k=0}^{i-1} 2^k T_s\right) + \left(\sum_{k=0}^{j-1} 2^k T_s\right) \\ &= RTT + (2^i - 1)T_s + (2^j - 1)T_s \\ &= RTT + (2^i + 2^j - 2)T_s \end{aligned} \tag{2}$$

The probability that L_h, the overall latency for a three-way handshake episode, is t seconds or less is:

$$P[L_h \leq t] = \sum_{L_h(i,j) \leq t} P_h(i,j) \tag{3}$$

Most TCP implementations abort connection establishment attempts after 4-6 failures. For loss rates low enough that most handshakes succeed before TCP gives up, it can be shown that (4) is a good approximation for the expected handshake time:

$$E[L_h] = RTT + T_s\left(\frac{1-p_r}{1-2p_r} + \frac{1-p_f}{1-2p_f} - 2\right) \tag{4}$$

This model assumes the TCP implementation complies with the TCP specification [37]. It does not model non-compliant implementations, such as current versions of Linux 2.2, that achieve slightly better performance by responding to retransmitted SYN segments with retransmitted SYN/ACK segments.

D. Data Transfer

As defined here, a data transfer begins when an application places data in its send buffer and ends when TCP receives an acknowledgment for the last byte in the buffer. We assume that during this transfer the sending application places data in the send buffer quickly enough that the sending TCP can send as fast as its window allows.

We decompose the data transfer latency, $E[T]$, for d data segments into four aspects: the initial slow start phase, the resulting packet loss (if any), the transfer of any remaining data, and the added delay from delayed acknowledgments. We begin by calculating the amount of data we expect to send in the initial slow start phase before encountering a packet loss or finishing the data transfer. From this we can deduce the time spent in slow start, the final congestion window in slow start, and thus the expected cost of loss recovery, if any. Then we use the steady-state throughput from [33] to approximate the cost of sending the remaining data, if any. Finally, we add any extra cost from delayed ACKs. We discuss each of these aspects in turn.

D.1 Initial Slow Start

We assume that the transfer is either the first transfer of a connection, or a later transfer on a connection that has experienced no losses yet. Under these circumstances, TCP begins in slow start mode, where it quickly increases its congestion window, *cwnd*, until it detects a packet loss.

$E[T_{ss}]$, the expected latency for the initial slow start phase, depends on the structure of the slow start episode. There are two important cases. In the first case, the sender's *cwnd* grows continuously until it detects a packet loss. In the second case, the sender's *cwnd* is eventually bounded by a maximum window, W_{max}, imposed by sender or receiver buffer limitations. To determine which case is appropriate, we need to calculate $E[d_{ss}]$, the number of data segments we expect the sender to send before losing a segment. From this we can deduce $E[W_{ss}]$, the window we would expect TCP to achieve at the end of slow start, were there no maximum window constraint. If $E[W_{ss}] \leq W_{max}$, then the window limitation has no effect, and $E[T_{ss}]$ is simply the time for a sender to send $E[d_{ss}]$ in the exponential growth mode of slow start. On the other hand, if $E[W_{ss}] > W_{max}$ then $E[T_{ss}]$ is the time for a sender to slow start up to $cwnd = W_{max}$ and then send the remaining data segments at a rate of W_{max} segments per round.

First we calculate $E[d_{ss}]$, the number of data segments we expect to send in the initial slow start phase before a loss occurs (not including the lost segment). Let p be the data segment loss rate. If $p = 0$, we expect to be able to send all d segments in slow start, so $E[d_{ss}] = d$. On the other hand, if $p > 0$, and we assume that the loss rate is independent of sender behavior, then

$$\begin{aligned} E[d_{ss}] &= \left(\sum_{k=0}^{d-1} (1-p)^k \cdot p \cdot k\right) + (1-p)^d \cdot d \\ &= \frac{(1-(1-p)^d)(1-p)}{p} + 1 \end{aligned} \tag{5}$$

Next we deduce the time spent in slow start. During slow start, as always, each round the sender sends as many data segments as its *cwnd* allows. Since the receiver sends one ACK for every b-th data segment that it receives, each round the sender will get approximately $cwnd/b$ ACKs. Because the sender is in slow start, for each ACK it receives, it increases its *cwnd* by one segment. Thus, if we use $cwnd_i$ to denote the sender's congestion window at the beginning of round i and, following [1], use γ to denote the rate of exponential growth of *cwnd* during slow start , we have:

$$\begin{aligned} cwnd_{i+1} &= cwnd_i + cwnd_i/b \\ &= (1 + 1/b) \cdot cwnd_i \\ &= \gamma \cdot cwnd_i \end{aligned} \tag{6}$$

If a sender starts with an initial *cwnd* of w_1 segments, then $ssdata_i$, the amount of data sent by the end of slow start round

i, can be closely approximated by a geometric series as

$$ssdata_i = w_1 + w_1 \cdot \gamma + w_1 \cdot \gamma^2 + \cdots + w_1 \cdot \gamma^{i-1} \quad (7)$$

$$= w_1 \cdot \frac{\gamma^i - 1}{\gamma - 1} \quad (8)$$

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

$$i = \log_\gamma \left(\frac{ssdata_i(\gamma - 1)}{w_1} + 1 \right) \quad (9)$$

From (7) and (9) it follows that $W_{ss}(d)$, the window TCP achieves after sending d segments in unconstrained slow start, is

$$W_{ss}(d) = w_1 \cdot \left(\frac{d(\gamma - 1)}{w_1} + 1 \right) \cdot \gamma^{-1}$$

$$= \frac{d(\gamma - 1)}{\gamma} + \frac{w_1}{\gamma} \quad (10)$$

Given typical parameters of $\gamma = 1.5$ and $1 \leq w_1 \leq 3$, equation (10) implies that $W_{ss}(d) \approx \frac{d}{3}$. Put another way, to reach any congestion window, w, a flow needs to send approximately $3w$. Interestingly, this implies that to reach full utilization for a bandwidth-delay product like 1.5Mbps · 70ms = 13KBytes, a TCP flow will need to transfer 39KBytes, a quantity larger than most wide-area TCP flows transfer. From this it is easy to see why many Internet flows spend most of their lifetimes in slow start, as observed in [3].

From (5) and (10) we can calculate the window size we would expect to have at the end of slow start, if we were not constrained by W_{max}:

$$E[W_{ss}] = \frac{E[d_{ss}](\gamma - 1)}{\gamma} + \frac{w_1}{\gamma} \quad (11)$$

so we can now determine whether we expect *cwnd* to be constrained by W_{max} during slow start.

If $E[W_{ss}] > W_{max}$ then slow start proceeds in two phases. First, *cwnd* grows up to W_{max}; from (10) the flow will send

$$d_1 = \frac{\gamma W_{max} - w_1}{\gamma - 1} \quad (12)$$

segments during this phase. From (9), this will take

$$r_1 = \log_\gamma \left(\frac{W_{max}}{w_1} \right) + 1 \quad (13)$$

rounds. During the second phase, the flow sends the remaining data at a rate of W_{max} packets per round, which will take

$$r_2 = \frac{1}{W_{max}} (d_{ss} - d_1) \quad (14)$$

rounds.

Combining (12), (13), and (14) for the case where when $E[W_{ss}] > W_{max}$, and using (9) for the simpler case where $E[W_{ss}] \leq W_{max}$, the time to send $E[d_{ss}]$ data segments in slow start is approximately

$$E[T_{ss}] = \begin{cases} RTT \cdot \left[\log_\gamma \left(\frac{W_{max}}{w_1} \right) + 1 + \right. \\ \left. \quad \frac{1}{W_{max}} \left(E[d_{ss}] - \frac{\gamma W_{max} - w_1}{\gamma - 1} \right) \right] \\ \quad \text{when } E[W_{ss}] > W_{max} \\ RTT \cdot \log_\gamma \left(\frac{E[d_{ss}](\gamma - 1)}{w_1} + 1 \right) \\ \quad \text{when } E[W_{ss}] \leq W_{max} \end{cases} \quad (15)$$

D.2 The First Loss

For some TCP flows, the initial slow start phase ends with the detection of a packet loss. Since slow start ends with a packet loss if and only if a flow has at least one loss, the probability of this occurrence is:

$$l_{ss} = 1 - (1 - p)^d \quad (16)$$

There are two ways that TCP detects losses: retransmission timeouts (RTOs) and triple duplicate ACKs. [34] gives a derivation of the probability that a sender in congestion avoidance will detect a packet loss with an RTO, as a function of packet loss rate and window size. They denote this probability by $Q(p, w)$:

$$Q(p, w) = \min \left(1, \frac{(1 + (1 - p)^3(1 - (1 - p)^{w-3}))}{(1 - (1 - p)^w)/(1 - (1 - p)^3)} \right) \quad (17)$$

The probability that a sender will detect a loss via triple duplicate ACKs is simply $1 - Q(p, w)$. Although $Q(p, w)$ was derived under the assumption that the sender is in congestion avoidance mode and has an unbounded amount of data to send, our experience has shown that it is equally applicable to slow start and senders with a limited amount of data. This is largely because $Q(p, w)$ is quite insensitive to the rate of growth of *cwnd*, and senders with a limited amount of data are already at high risk for RTOs because they will usually have small windows. In practice, we suspect that the fast recovery strategy used by the sender has a far greater impact; senders using SACK, FACK, or NewReno should be able to achieve the behavior predicted by $Q(p, w)$, while Reno senders will have difficulty achieving this performance when they encounter multiple losses in a single round [26, 27].

The expected cost of an RTO is also derived in [34]:

$$E[Z^{TO}] = \frac{G(p)T_0}{1 - p} \quad (18)$$

where T_0 is the average duration of the first timeout in a sequence of one or more successive timeouts, and $G(p)$ is given by:

$$G(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6 \quad (19)$$

The expected cost of a fast recovery period depends on the number of packets lost, the fast recovery strategy of the sender's TCP implementation, and whether the receiving TCP implementation is returning selective acknowledgment (SACK) options. In the best case, where there is a single loss or the sender can use SACK information, fast recovery often takes only a single RTT; in the worst case, NewReno will require one RTT for each lost packet. Our experience indicates that which of these possibilities actually occurs is usually not important to the model's predictions, so, as in the model of [34], for the sake of simplicity, we assume that fast recovery always takes a single RTT.

Combining these results, the expected cost for any RTOs or fast recovery that happens at the end of the initial slow start phase is:

$$T_{loss} = l_{ss} \cdot \left(Q(p, E[W_{ss}]) \cdot E[Z^{TO}] + (1 - Q(p, E[W_{ss}])) \cdot RTT \right) \quad (20)$$

D.3 Transferring the Remainder

In order to approximate the time spent sending any data remaining after slow start and loss recovery, we estimate the amount of remaining data and apply the steady-state model from [34].

The amount of data left after slow start and any following loss recovery is approximately

$$E[d_{ca}] = d - E[d_{ss}] \tag{21}$$

This is only an approximation, because the actual amount of data remaining will also depend on where in the window the loss occurs, how many segments are lost, the size of the congestion window at the time of loss, W_{max}, and the recovery algorithm. However, since the model seems accurate in most cases even with this simplification, for the sake of simplicity we use Equation (21).

When there are $E[d_{ca}] > 0$ segments left to send, we approximate the time to transfer this data using the steady-state throughput model from [33]. This model gives throughput, which we will denote $R(p, RTT, T_0, W_{max})$, as a function of loss rate, p, round trip time, RTT, average RTO, T_0, and maximum window constraint, W_{max}:

$$R = \begin{cases} \dfrac{\frac{1-p}{p} + \frac{W(p)}{2} + Q(p,W(p))}{RTT(\frac{b}{2}W(p)+1) + \frac{Q(p,W(p))G(p)T_0}{1-p}} & \text{if } W(p) < W_{max} \\ \dfrac{\frac{1-p}{p} + \frac{W_{max}}{2} + Q(p,W_{max})}{RTT(\frac{b}{8}W_{max} + \frac{1-p}{pW_{max}} + 2) + \frac{Q(p,W_{max})G(p)T_0}{1-p}} & \text{otherwise} \end{cases} \tag{22}$$

where $Q(p, w)$ is given in (17), $G(p)$ is given in (19), and $W(p)$ is the expected congestion window at the time of loss events when in congestion avoidance, also from [33]:

$$W(p) = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{23}$$

Using these results for the expected throughput, we approximate the expected time to send the remaining data, $E[d_{ca}] > 0$, as

$$E[T_{ca}] = E[d_{ca}]/R(p, RTT, T_0, W_{max}) \tag{24}$$

Using a model for steady-state throughput to characterize the cost of transferring the remaining data introduces several errors.

First, when the sender detects a loss in the initial slow start phase, its *cwnd* will often be much larger than the steady-state average *cwnd*. Combining (10) and the analysis of [27], the sender will have to detect roughly $\log_2 \frac{1/3p}{\sqrt{3/(2bp)}}$ loss indications to bring *cwnd* from its value at the end of slow start, $1/3p$, to its steady state value, $\sqrt{3/(2bp)}$. For loss rates of 5% and higher, the sender exits slow start at nearly the steady-state window value, so the error in our approach should be small. For loss rates of 0.1% and below, it can take three or more loss indications – corresponding to megabytes of data – to reach steady state, so our approach will often overestimate the latency of such transfers.

Another source of error derives from the fact that (22) does not model slow start after retransmission timeouts (RTOs). For loss rates above 1%, the error this introduces should be small, since for these loss rates congestion avoidance has throughput that is similar to that of slow start after RTOs. For lower loss rates, RTOs should be uncommon. However, when they occur, their long delays may overwhelm the details of *cwnd* growth.

Finally, RTO durations vary widely. Using an average RTO to model the duration of a specific short TCP flow will introduce significant error.

D.4 Delayed Acknowledgments

Delayed acknowledgments comprise the final component of TCP latency that we consider in our model. There are a number of circumstances under which delayed acknowledgments can cause relatively large delays for short transfers. The most common delay occurs when the sender sends an initial *cwnd* of 1 MSS. In this case the receiver waits in vain for a second segment, until finally its delayed ACK timer fires and it sends an ACK. In BSD-derived implementations this delay is uniformly distributed between 0ms and 200ms, while in Windows 95 and Windows NT 4.0 this delay is distributed uniformly between 100ms and 200ms. Delayed ACKs may also lead to large delays when the sender sends a small segment and then the Nagle algorithm prevents it from sending further segments [17, 29], or when the sender sends segments that are not full-sized, and the receiver implementation waits for two full-sized segments before sending an ACK.

For our simulations and measurements, when senders use an initial *cwnd* of 1 MSS we model the expected cost of the first delayed ACK, which we denote $E[T_{delack}]$, as the expected delay between the reception of a single segment and the delayed ACK for that segment – 100ms for BSD-derived stacks, and 150ms for Windows.

D.5 Combining the Results

To model the expected time for data transfer, we use the sum of the expected delays for each of the components, including (15), (20), (24), and the delayed ACK cost:

$$E[T] = E[T_{ss}] + E[T_{loss}] + E[T_{ca}] + E[T_{delack}] \tag{25}$$

E. Modeling Distributions

The model as given in (25) is a prediction of latency given the particular parameters experienced by a particular flow. In our experience, for a set of transfers of the same size over the same high bandwidth-delay path, the most important determinants of overall latency are the number of losses, the average timeout duration, and the cost of delayed ACKs. As a result, to approximate the distribution of latency for a set of flows, one can consider the range of possible loss rates and delayed ACK costs, and estimate the likelihood of each scenario, along with the latency expected with that scenario. In section IV-A.2 we apply this method to simulations using a Bernoulli loss model and delayed ACK costs uniformly distributed between 0 and 200ms.

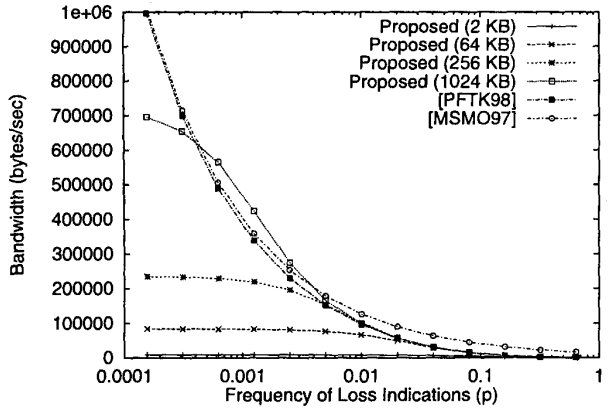

Fig. 2. Comparing the throughput predicted by the steady state models to the throughput predicted by our proposed model for varying transfer sizes. Here $MSS = 1460$ bytes, $RTT = 100$ ms, $w_1 = 1$ segment, $\gamma = 1.5, T_0 = 1$ sec, $W_{max} = 10$ MBytes.

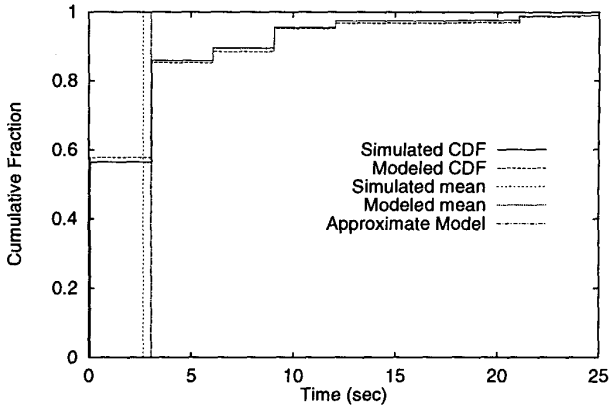

Fig. 3. Distribution and mean of connection establishment times from the model (3), the approximate model (4), and 1000 ns simulations with $p_f = 0.3$, $p_r = 0.2$.

F. Comparison with Earlier Models

Our proposed model, (25), is a generalization of two previous approaches. In the case where there are no packet losses, (25) reduces to (15), a model for the time to send d segments in slow start mode. This special case corresponds closely to the simpler models derived in [18, 24]. In the case where d is very large, the total time given by (25) is dominated by (24), the time to transfer data after the first loss. In this case, the behavior corresponds very closely with the underlying throughput model, (22), from [34].

Figure 2 explores the relationship between our proposed model and the models of [27] and [34]. It gives the throughput predicted by the proposed model, (25), for each of several transfer sizes, as well as the steady-state throughputs predicted by the expression $\sqrt{3/4} \cdot MSS/(RTT\sqrt{p})$, from [27], and (22), from [34]. As mentioned earlier, when there is at least one expected loss, our proposed model agrees closely with [34], which has been shown to work well for flows that suffer even a few losses [5]. On the other hand, when there are no losses, the proposed model predicts that short flows suffer because they do not have time to reach a steady-state *cwnd*, whereas long flows will do well because their *cwnd* grows beyond its steady-state value.

III. Verifying the Connection Establishment Model

Figure 3 compares the mean and distribution of connection establishment times given by (3) and the mean given by the approximate model (4) to the mean and distribution for 1000 ns [31] simulations with $RTT = 70$ ms and Bernoulli packet losses with $p_f = 0.3$ and $p_r = 0.2$. These simulations used the FullTCP implementation, modeled closely after the 4.4BSD TCP implementation. Both the model and the approximation fit well. Results are similar for other scenarios with both p_f and p_r well below 0.5.

Figure 4 summarizes the performance of the full and approximate model, comparing them against 1000 ns trials in scenarios with $p_r = 0.0$ and $0 \leq p_f \leq 0.45$. The full model fits well across these simulations, but the approximate model diverges sharply as p_f approaches 0.5, where its assumption of unbounded wait times fails.

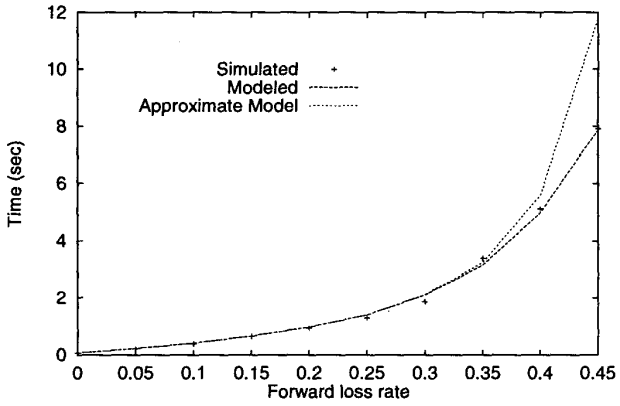

Fig. 4. Expected connection establishment latency from ns simulations (1000 trials at each loss rate), the model (3), and the approximate model (4).

IV. Verifying the Data Transfer Model

A. Simulations

A.1 Flows Without Loss

For simulated TCP flows that do not suffer packet loss, the expression (15) describes TCP behavior very closely, as Figure 5 shows. This figure depicts the simulated and modeled latency for 2,376 TCP transfers simulated in ns. In each simulation, a FullTCP sender transfers data over a 1Gbit link to a FullTCP receiver. The link buffers were provisioned to prevent packet loss. The trials consisted of the cross-product of $w_1 = \{2, 3, 4\}$ segments, $\gamma = \{1.5, 2\}$ (with and with-

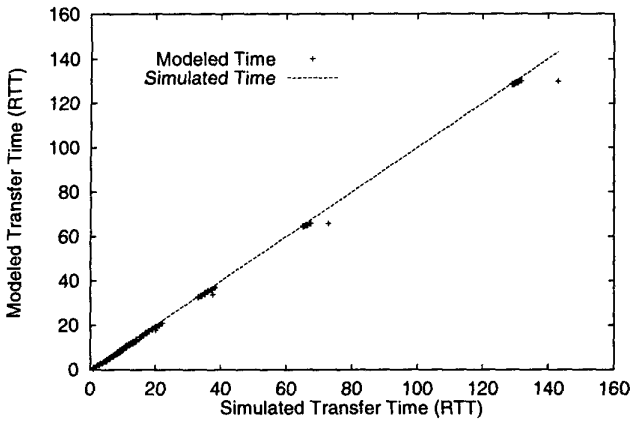

Fig. 5. The simulated latency for 2,376 TCP data transfers that experienced no packet loss, compared with the modeled latency from (25) (or, equivalently, (15)).

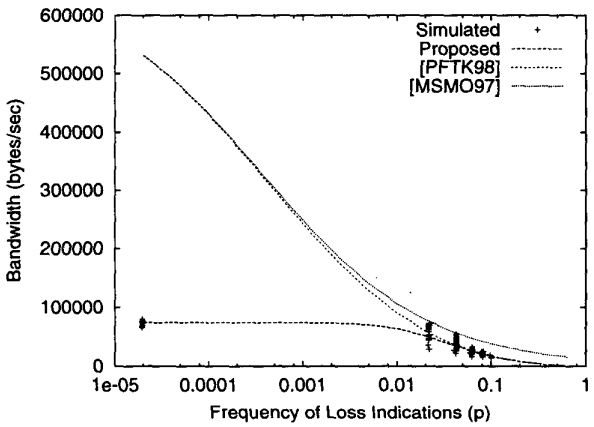

Fig. 6. Scatter plot of simulated performance with model predictions overlaid. These were 64 KByte transfers with $MSS = 1460$ bytes, W_{max} of 4MBytes, $w_1 = 1$ segment, $\gamma = 1.5$, $RTT = 100$ ms, $T_0 = 519$ ms, $p_f = 0.05$ and $p_r = 0$.

out delayed ACKs), $d = \{1, 2, 4, \ldots 1024\}$ segments, $MSS = \{536, 1460, 4312\}$ bytes, $W_{max} = \{8, 32, 128, 512\}$ segments, and $RTT = \{16, 64, 256\}$ ms. The model agrees quite closely with the simulations; the average error is 0.69 RTTs, and the average relative error is 22%. The three outliers at 37, 72, and 143 RTTs correspond to trials with a window of just 8 segments, where throughput was hurt because the 200ms delayed ACK timer of the recipient was mis-aligned with the 256ms RTT ACK-clocking employed by the sender.

A.2 Flows Suffering Losses

Figures 6 and 7 illustrate how well [34] and (25) match the performance of flows that suffer moderate-to-high levels of loss. Figure 6 shows a scatter plot depicting the bandwidth and loss rate experienced by each of the 100 simulated `FullTCP` flows, with the model predictions overlaid. Each flow transferred

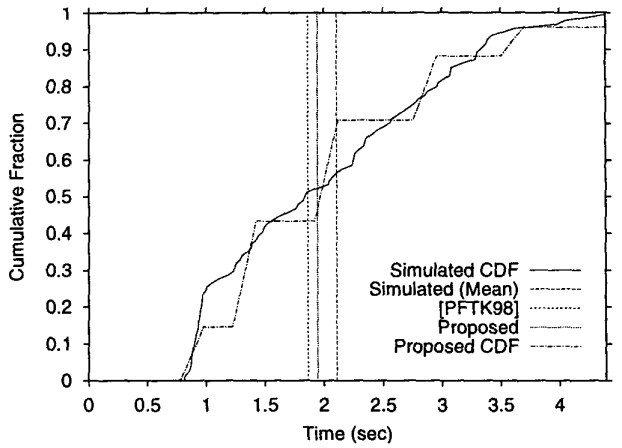

Fig. 7. The distribution and mean of latencies from the experiment described in Figure 6.

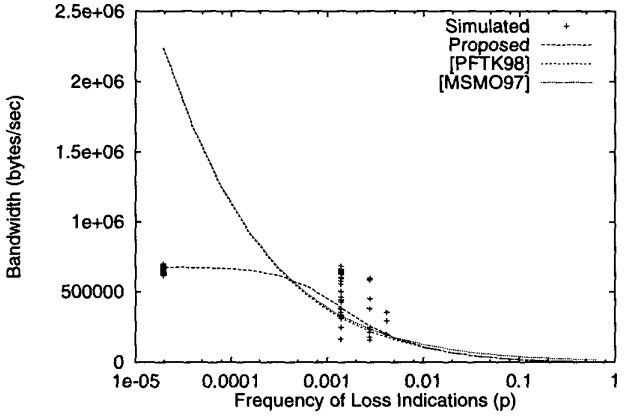

Fig. 8. Scatter plot of simulated performance with model predictions overlaid. These were 1 MByte transfers with $MSS = 1460$ bytes, W_{max} of 4MBytes, $w_1 = 1$ segment, $\gamma = 1.5$, $RTT = 100$ ms, $T_0 = 450$ ms, $p_f = 0.001$ and $p_r = 0$.

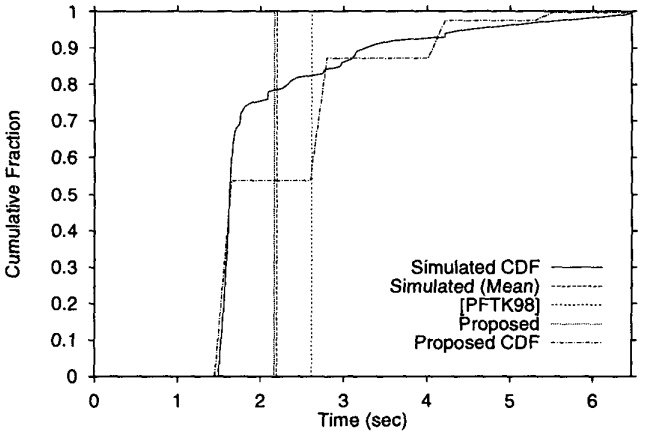

Fig. 9. The distribution and mean of latencies from the experiment described in Figure 8.

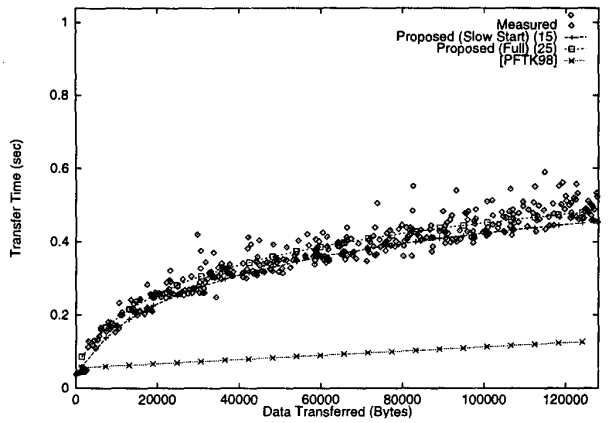

Fig. 10. Measured and modeled latencies for 403 transfers from the University of Washington to the UC-Davis.

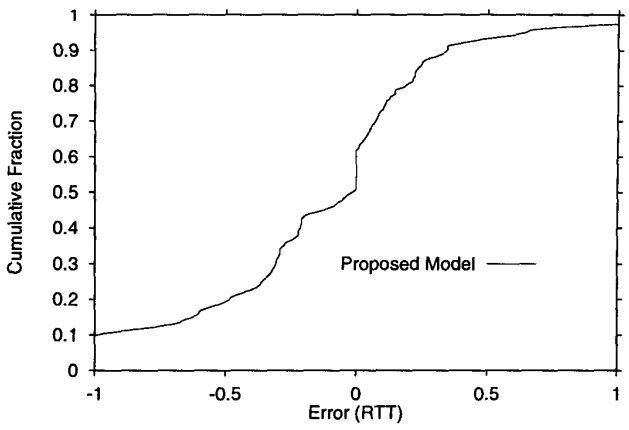

Fig. 11. The error, (modeled – measured)/RTT, between the proposed model, (25), and the HTTP measurements, for all 33,208 flows (97%) that suffered no packet losses. Note that this is RTT-normalized error, so the model is within 1 RTT of the actual time for 85% of flows.

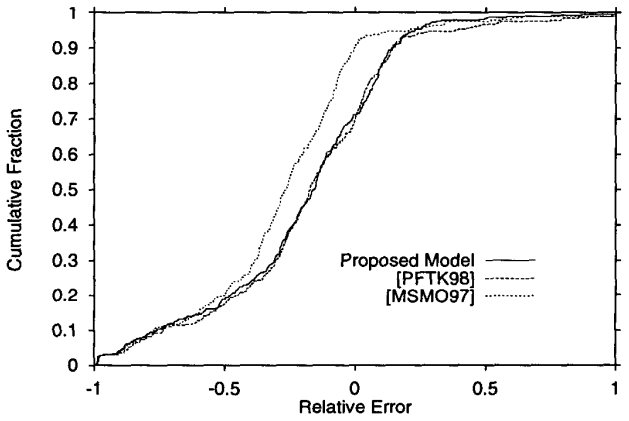

Fig. 12. The relative error between the models and the HTTP measurements, for all 357 flows (1%) that suffered triple duplicate ACKs but no RTOs.

64KBytes over a path with synthetically-generated Bernoulli losses with an average loss rate of $p_f = 0.05$ and $p_r = 0$. The proposed model, (25) fits the trials that experience no packet loss, while both [34] and (25) provide a reasonable fit to those trials that do experience loss. Figure 7 shows the distribution of latencies for these trials. Both [34] and (25) capture the average latency, and the modeled distribution, derived using the technique described in Section II-E, provides a reasonable characterization of the distribution of latencies. There is considerable variance in the latency in this case, with 25% of flows completing in half the average time, and 25% of flows taking half again as long as the average time. In our experience, this technique yields a good approximation to the latency distribution whenever there are enough packet losses for [34] to provide a good fit for the average latency.

Figures 8 and 9 provide the corresponding view of long transfers (1 MByte) over paths with low loss rates. The proposed model, (25), captures the average latency as well as the latency experienced by the half of flows that see no loss. However, neither [34] nor (25) predicts the performance seen by flows that experience a single loss, as these flows enter congestion avoidance with a *cwnd* far larger than the steady-state value. We have preliminary results characterizing the dynamics of *cwnd* as it converges to the steady-state value after slow start. It should be possible to capture aspects of the behavior of these long flows that suffer only a few losses by using an approach along these lines.

B. Controlled Internet Measurements

In order to examine how well our proposed model, (25), fits TCP behavior in the Internet, we performed a number of TCP transfers from a Linux 2.0.35 sender at the University of Washington to other Internet sites. Figure 10 shows an example. It depicts the latency of 403 transfers of varying sizes to the University of California at Davis, together with the predictions from (25) and [34], using the average packet loss rate across all trials. Since the average loss rate was only 0.02%, and as a consequence most flows suffered no losses, (25) fits quite well, whereas [34] does not.

C. Live HTTP Measurements

In order to undestand how well the various TCP models describe typical TCP data transfers, we compared them to a set of HTTP traces. These traces consist of client-side packet-level traces of 34,318 TCP flows transporting single HTTP GET operations made from `wget` web clients at three well-connected U.S. universities – the University of Washington-Seattle, the University of California-Berkeley, and Duke University – to 50 web servers spread throughout the US. Twenty-five of the servers were chosen from the web's most popular sites, as determined by the analysis of web proxy logs [19]. The remaining 25 servers were chosen at random from Yahoo!'s database of web sites [43]. Each HTTP GET operation fetched the `index.html` object from the given site; the average size of

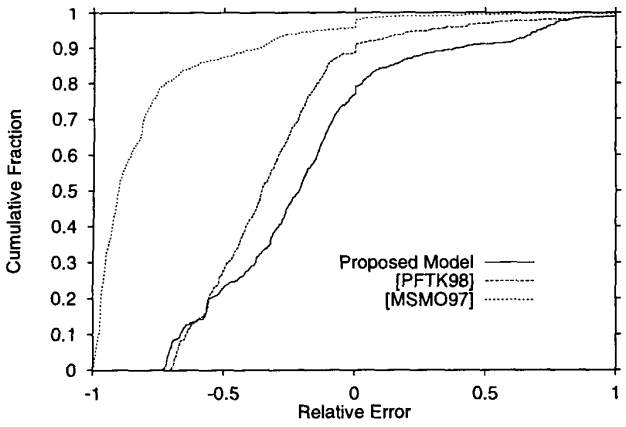

Fig. 13. The relative error between the models and the HTTP measurements, for all 753 flows (2%) that suffered RTOs.

these objects was 13.5KBytes.

For each flow, we extracted the total data transfer time, defined as the period starting when the first data segment arrived at the client and ending when last data segment arrives at the client, minus the delay due to delayed ACKs. We also extracted the MSS, the initial window size, and estimated the RTT using the receiver-oriented techniques described in [1]. We estimate the number of fast retransmit events by observing data segments arriving after corresponding triple-duplicate ACKs. We estimate the number and average duration of timeouts by using several heuristics, including the arrival of duplicate data segments, the arrival of data segments that fill a sequence space hole more than 200ms old, and the arrival of in-order data segments that arrive after an idle period of one second or more.

We split these flows into three classes, based on the character of the losses they suffered. For each flow, we examine the error between the measured latency and the predictions made by the models from [27], [34], and (25), using the actual d, MSS, W_{max}, w_1, RTT, p, and T_0 observed by that flow.

Figure 11 shows the RTT-normalized error in latency predictions, (modeled − measured)/RTT, for those flows that suffered no packet loss. We show only the proposed model, (25), since the models from [27] and [34] are undefined for $p = 0$. More than 85% of the model's predictions are within one RTT of the actual transfer times. Turning to the relative error, (modeled − measured)/measured, the median relative error for these cases is .16, meaning for 50% of flows that suffered no loss, (25) is within 16% of the actual transfer latency.

Figure 12 shows the relative error in latency predictions for those flows that suffered losses but were able to recover from all losses using fast retransmit and fast recovery. All three models provide similar predictions. Similarly, Figure 13 shows the relative error in latency predictions for the flows that suffered retransmission timeouts. The [27] model fits poorly, as it assumes that timeouts do not occur. The (25) and [34] models also have significant error. Note that, once again, in the presence of packet loss, (25) and [34] provide similar predictions.

V. Conclusion

In this paper we presented new models for TCP connection establishment and TCP slow start. We used these models to extend the steady-state model from [34], which assumes at least one packet loss, to characterize the latency of TCP flows that suffer no packet losses. Using simulation and measurement, we found that the connection establishment model seems promising, and that the new, extended data transfer model characterizes flows of varying lengths under varying loss conditions.

Furthermore, we described a technique for estimating the distribution of latencies for TCP transfers and showed simulations suggesting that this method can approximate the often wide distribution of data transfer latencies under a range of conditions.

Acknowledgments

We would like to thank Arnold Kim for suggesting a technique to derive (4), Sally Floyd and Neil Spring for providing comments on earlier drafts of this paper, Jitendra Padhye for providing analysis scripts, Robert Morris for providing traces, and John Zahorjan and Anna Karlin for providing advice and encouragement.

References

[1] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. In *SIGCOMM '99*, August 1999.

[2] Mark Allman, Vern Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

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

[4] Hari Balakrishnan, Mark Stemm, Srinivasan Seshan, and Randy H. Katz. Analyzing stability in wide-area network performance. In *SIGMETRICS '97*, June 1997.

[5] Juerg Bolliger, Thomas Gross, and Urs Hengartner. Bandwidth modelling for network-aware applications. In *INFOCOM '99*, March 1999.

[6] J. Bolot and T. Turletti. Experience with rate control mechanisms for packet video in the Internet. *Computer Communications Review*, 28(1), January 1998.

[7] J. Bolot and A. Vega-Garcia. Control mechanisms for packet audio in the Internet. In *INFOCOM '96*, March 1996.

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

[9] Claudio Casetti and Michela Meo. A new approach to model the stationary behavior of TCP connections. In *INFOCOM 2000*, March 2000.

[10] K. Claffy, Greg Miller, and Kevin Thompson. The nature of the beast: Recent traffic measurements from an Internet backbone. In *Proceedings of INET '98*, July 1998.

[11] 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.

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

[13] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[14] Sally Floyd and Tom Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.

[15] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[16] Steven D. Gribble and Eric A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *USITS '97*, December 1997.

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

[18] 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.
[19] `http://www.100hot.com/`.
[20] Van Jacobson. Congestion avoidance and control. *SIGCOMM '88*, August 1988.
[21] 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.
[22] 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.
[23] Bruce A. Mah. An empirical model of HTTP network traffic. In *INFOCOM '97*, April 1997.
[24] Jamshid Mahdavi. TCP performance tuning. `http://www.psc.edu/networking/tcptune/slides/`, April 1997.
[25] Sam Manthorpe. *Implications of the Transport Layer for Network Dimensioning*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 1997.
[26] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *SIGCOMM '96*, August 1996.
[27] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.
[28] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement options. RFC 2018, April 1996.
[29] Greg Minshall, Yasushi Saito, Jeffrey C. Mogul, and Ben Verghese. Application performance pitfalls and TCP's Nagle algorithm. In *Workshop on Internet Server Performance*, May 1999.
[30] Archan Misra and Teunis Ott. The window distribution of idealized TCP congestion avoidance with variable packet loss. In *INFOCOM '99*, March 1999.
[31] UCB/LBNL/VINT network simulator - ns (version 2).
[32] Teunis J. Ott, T. V. Lakshman, and Larry H. Wong. SRED: Stabilized RED. In *INFOCOM '99*, March 1999.
[33] Jitendra Padhye, Victor Firoiu, and Don Towsley. A stochastic model of TCP Reno congestion avoidance and control. Technical Report 99-02, University of Massachusetts, 1999.
[34] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM '98*, September 1998.
[35] Craig Partridge and Timothy J. Shepard. TCP/IP performance over satellite links. *IEEE Network*, pages 44–49, September/October 1997.
[36] Vern Paxson. End-to-end Internet packet dynamics. In *SIGCOMM '97*, September 1997.
[37] Jon Postel, Editor. Transmission Control Protocol — DARPA Internet Program Protocol Specification. RFC 793, September 1981.
[38] Lili Qiu, Yin Zhang, and S. Keshav. On individual and aggregate TCP performance. Technical Report TR99-1744, Cornell University, May 1999.
[39] Stefan Savage. Sting: a TCP-based network measurement tool. In *USITS '99*, October 1999.
[40] `http://www.ens.fr/~mistral/tcpworkshop.html`, December 1998.
[41] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6), November 1997.
[42] L. Vivisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *INFOCOM '98*, April 1998.
[43] Yahoo! Inc. Random Yahoo! Link. `http://random.yahoo.com/bin/ryl`.
[44] Maya Yajnik, Sue Moon, Jim Kurose, and Don Towsley. Measurement and modelling of the temporal dependence in packet loss. In *INFOCOM '99*, March 1999.