The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm

Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi *
<mathis@psc.edu> <semke@psc.edu> <mahdavi@psc.edu>
Pittsburgh Supercomputing Center

Teunis Ott
<tjo@bellcore.com>
Bellcore

Abstract

In this paper, we analyze a performance model for the TCP Congestion Avoidance algorithm. The model predicts the bandwidth of a sustained TCP connection subjected to light to moderate packet losses, such as loss caused by network congestion. It assumes that TCP avoids retransmission timeouts and always has sufficient receiver window and sender data. The model predicts the Congestion Avoidance performance of nearly all TCP implementations under restricted conditions and of TCP with Selective Acknowledgements over a much wider range of Internet conditions.

We verify the model through both simulation and live Internet measurements. The simulations test several TCP implementations under a range of loss conditions and in environments with both drop-tail and RED queuing. The model is also compared to live Internet measurements using the TReno diagnostic and real TCP implementations.

We also present several applications of the model to problems of bandwidth allocation in the Internet. We use the model to analyze networks with multiple congested gateways; this analysis shows strong agreement with prior work in this area. Finally, we present several important implications about the behavior of the Internet in the presence of high load from diverse user communities.

1 Introduction

Traffic dynamics in the Internet are heavily influenced by the behavior of the TCP Congestion Avoidance algorithm [Jac88a, Ste97]. This paper investigates an analytical performance model for this algorithm. The model predicts end-to-end TCP performance from properties of the underlying IP path. This paper is a first step at discovering the relationship between end-to-end application performance, as observed by an Internet user, and hop-by-hop IP performance, as might be monitored and marketed by an Internet Service Provider.

Our initial inspiration for this work was the "heuristic analysis" by Sally Floyd [Flo91].

This paper follows a first principles derivation of the stationary distribution of the congestion window of ideal TCP Congestion Avoidance subject to independent congestion signals with constant probability. The derivation, by Teunis Ott, was presented at DIMACS [OKM96b] and is available on line [OKM96a]. The full derivation and formal analysis is quite complex and is expected to appear in a future paper.

We present a simple approximate derivation of the model, under the assumption that the congestion signal losses are periodic. This arrives at the same mathematical form as the full derivation, although the constant of proportionality is slightly different. This paper is focused on evaluating the model's applicability and impact to the Internet.

The model applies whenever TCP's performance is determined solely by the Congestion Avoidance algorithm (described below). We hypothesize that it applies to nearly all implementations of SACK TCP (TCP with Selective Acknowledgements) [MMFR96] under most normal Internet conditions and to Reno TCP [Jac90, Ste94, Ste97] under more restrictive conditions. To test our hypothesis we examine the performance of the TCP Congestion Avoidance algorithm in three ways. First, we look at several TCP implementations in a simulator, exploring the performance effects of random packet loss, packet loss due to drop-tail queuing, phase effects [FJ92], and Random Early Detection (RED) queuing [FJ93]. Next, we compare the model to Internet measurements using results from the TReno ("tree-no") [Mat96] user mode performance diagnostic. Finally, we compare the model to measurements of packet traces of real TCP implementations.

Many of our experiments are conducted with an updated version of the FACK TCP [MM96a], designed for use with Selective Acknowledgements. We call this Forward Acknowledgments with Rate-Halving (FACK-RH) [MM96b]. Except as noted, the differences between FACK-RH and other TCP implementations do not have significant effects on the results. See Appendix A for more information about FACK-RH.

*This work is supported in part by National Science Foundation Grant No. NCR-9415552.

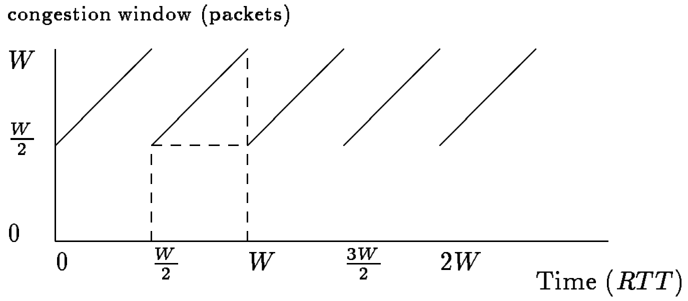

Figure 1: TCP window evolution under periodic loss
Each cycle delivers $(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = 1/p$ packets and takes $W/2$ round trip times.

2 The Model

The TCP Congestion Avoidance algorithm [Jac88a] drives the steady-state behavior of TCP under conditions of light to moderate packet losses. It calls for increasing the congestion window by a constant amount on each round trip and for decreasing it by a constant multiplicative factor on each congestion signal.[1] Although we assume that congestion is signaled by packet loss, we do not assume that every packet loss is a new congestion signal. For all SACK-based TCPs, multiple losses within one round trip are treated as a single congestion signal. This complicates our measurements of congestion signals.

We can easily estimate TCP's performance by making some gross simplifications. Assume that TCP is running over a lossy path which has a constant round trip time (RTT) because it has sufficient bandwidth and low enough total load that it never sustains any queues. For ease of derivation, we approximate **random packet loss at constant probability** p by assuming that the link delivers approximately $1/p$ consecutive packets, followed by one drop. Under these assumptions the congestion window (*cwnd* in most implementations) traverses a perfectly periodic sawtooth. Let the maximum value of the window be W packets. Then by the definition of Congestion Avoidance, we know that during equilibrium, the minimum window must be $W/2$ packets. If the receiver is acknowledging every segment, then the window opens by one segment per round trip, so each cycle must be $W/2$ round trips, or $RTT * W/2$ seconds. The total data delivered is the area under the sawtooth, which is $(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = \frac{3}{8}W^2$ packets per cycle. By assumption, each cycle also delivers $1/p$ packets (neglecting the data transmitted during recovery). Solving for W we get:

$$W = \sqrt{\frac{8}{3p}} \qquad (1)$$

Substitute W into the bandwidth equation below:

$$BW = \frac{\textit{data per cycle}}{\textit{time per cycle}} = \frac{MSS * \frac{3}{8}W^2}{RTT * \frac{W}{2}} = \frac{MSS/p}{RTT\sqrt{\frac{2}{3p}}} \qquad (2)$$

Collect the constants in one term, $C = \sqrt{3/2}$, then we arrive at:

$$BW = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \qquad (3)$$

Other forms of this derivation have been published [Flo91, LM94] and several people have reported unpublished, "back-of-the-envelope" versions of this calculation [Mat94a, Cla96].

Derivation	ACK Strategy	C
Periodic Loss	Every Packet	$1.22 = \sqrt{3/2}$
(derived above)	Delayed	$0.87 = \sqrt{3/4}$
Random Loss	Every Packet	1.31
follows [OKM96a]	Delayed	0.93

Table 1: Derived values of C under different assumptions.

The **constant of proportionality** (C) lumps together several terms that are typically constant for a given combination of TCP implementation, ACK strategy (delayed vs non-delayed)[2], and loss mechanism. Included in the TCP implementation's contribution to C

[1] The window is normally opened at the constant rate of one **maximum segment size** (MSS) per **round trip time** (RTT) and halved on each congestion signal. In actual implementations, there are a number of important details to this algorithm.

Opening the congestion window at a constant rate is actually implemented by opening the window by small increments on each acknowledgment, such that if every segment is acknowledged, the window is opened by one segment per round trip. Let W be the window size in packets. Each acknowledgment adjusts the window: W += $1/W$, such that W acknowledgments later W has increased by 1. Since W equals $cwnd/MSS$, we have $cwnd$ += $MSS * MSS/cwnd$, which is how the window opening phase of congestion avoidance appears in the code.

When the congestion window is halved on a congestion signal, it is normally rounded down to an integral number of segments. In most implementations the window is never adjusted below some floor, typically 2 segments. Both derivations neglect rounding and this low window limit. [Flo91] considers rounding, resulting in a small correction term.

We are also neglecting the details of TCP data recovery and retransmission. Some form of Fast Retransmit and/or Fast Recovery, with or without SACK, is required. The important detail is that the loss recovery is completed in roughly one round trip time, TCP's Self-clock is preserved, and that the new congestion window is half of the old congestion window.

[2] The Delayed Acknowledgment ("DA") algorithm [Ste94] suppresses half of the TCP acknowledgments to reduce the number of tiny messages in the Internet. This changes the Congestion Avoidance algorithm because the window increase is driven by the returning acknowledgments. The net effect is that when the TCP receiver sends Delayed Acknowledgments, the sender only opens the window by $MSS/2$ on each round trip. This term can be carried through any of the derivations and always reduces C by $\sqrt{2}$.

The receiver always suppresses Delayed Acknowledgements when it holds partial data. During recovery the receiver acknowledges every incoming segment. The receiver also suppresses Delayed Acknowledgements (or more precisely, transmits acknowledgements on a timer) when the data packets arrive more than 200 ms apart.

There are also a number of TCP implementations which have are the constants used in the Congestion Avoidance algorithm itself.

The model is not expected to apply under a number of situations where pure Congestion Avoidance does not fully control TCP performance. In general these phenomenon reduce the performance relative to that which is predicted by the model. Some of these situations are:

1. If the data receiver is announcing too small a window, then TCP's performance is likely to be fully controlled by the receiver's window and not at all by the Congestion Avoidance algorithm.

2. Likewise, if the sender does not always have data to send, the model is not likely to apply.

3. The elapsed time consumed by TCP timeouts is not modeled. Many non-SACK TCP implementations suffer from timeouts when they experience multiple packet losses within one round trip time [Flo95, MM96a]. These TCP implementations do not fit the model in environments where they experience such losses.

4. TCP implementations which exhibit go-back-N behaviors do not attain the performance projected by the model because the model does not account for the window consumed by needlessly retransmitting data. Although we have not studied these situations extensively, we believe that Slow-start, either following a timeout or as part of a normal Tahoe recovery, has at least partially go-back-N behavior, particularly when the average window is small.

5. TCP implementations which use other window opening strategies (e.g. TCP Vegas [BOP94, DLY95]) will not fit the model.

6. In some situations, TCP may require multiple cycles of the Congestion Avoidance algorithm to reach steady-state[3]. As a result, short connections do not fit the model.

Except for Item 6, all of these situations reduce TCP's average throughput. Under many circumstances it will be useful to view Equation 3 as a bound on performance. Given that Delayed Acknowledgements are mandatory, C is normally less than 1. Thus in many practical situations, we can use a simpler bound:

$$BW < \left(\frac{MSS}{RTT}\right)\frac{1}{\sqrt{p}} \tag{4}$$

We will show that it is important that appropriate measurements be used for p and RTT. For example SACK TCP will typically treat multiple packet losses in one RTT as a single congestion signal. For this case, the proper definition for p is the **number of congestion signals per acknowledged packet**.

Although these derivations are for a rather restricted setting, our empirical results suggest that the model is more widely applicable.

3 Simulation

All of our simulations use the LBL simulator, "ns version 1", which can be obtained via FTP [MF95].

Most of the simulations in this paper were conducted using the topology in Figure 2. The simulator associates queuing properties (drop mechanism, queue size, etc.) with links. The nodes (represented by circles) implement TCP, and do not themselves model queues. We were careful to run the simulations for sufficient time to obtain good measures of TCP's average performance[4].

This single link is far too simple to model the complexity of a real path through the Internet. However, by manipulating the parameters (delay, BW, loss rate) and queuing models (drop-tail, RED) we will explore the properties of the performance model.

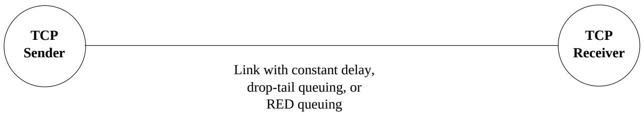

Figure 2: The simulation topologies

3.1 Queueless Random Packet Loss

In our first set of experiments, the single link in Figure 2 was configured to model the conditions under which Equation 3 was derived in [OKM96a]: constant delay and fixed random packet loss. These conditions were represented by a lossy, high bandwidth link[5] which does not sustain a queue.

The choice of our FACK-RH TCP implementation does not affect the results in this section, except that it is able to remain in Congestion Avoidance at higher loss rates than other TCPs. This phenomenon will be discussed in detail in Section 3.4. The receiver is using standard Delayed Acknowledgements.

The network was simulated for various combinations of delay, MSS, and packet loss. The simulation used three typical values for MSS: 536, 1460, and 4312 bytes. The one-way delay spanned five values from 3 ms to 300 ms; and the probability of packet loss was randomly selected across four orders of magnitude, spanning roughly from 0.00003 to 0.3 (uniformly distributed

bugs in their Delayed Acknowledgment algorithms such that they send acknowledgments less frequently than 1 per 2 data segments. These bugs further reduce C.

[3] This problem is discussed in Appendix B. All of the simulations in this paper are sufficiently long such that they unambiguously reach equilibrium.

[4] This was done by using the bandwidth-delay product to estimate an appropriate duration for the simulation, such that Congestion Avoidance experienced 50 or more cycles. The duration was confirmed from instruments in the simulator.

[5] In order to make sure that the link bandwidth was not a limiting factor, the link bandwidth selected was more than 10 times the estimated bandwidth required. We then confirmed that the link did not sustain a queue.

in $log(p)$). Since each loss was independent (and assumed to be relatively widely spaced), each loss was considered to be a congestion signal.

In Figure 3 we assume $C = 1$ and plot the simulation vs. the model. Each point represents one combination of RTT, MSS, and p in the simulation. The X axis represents the bandwidth estimated by the model from these measurements, while the Y axis represents the bandwidth as measured by the simulation. Note that the bandwidth, spanning nearly five orders of magnitude, has a fairly strong fit along one edge of the data. However, there are many outlying points where the simulation does not attain the predicted bandwidth.

In Figure 4 we plot a different view of the same data to better illuminate the underlying behaviors. Simulations that experienced timeouts are indicated with open markers. For the remainder of the figures (except where noted), we rescale the Y axis by RTT/MSS. The Y axis is then $BW * RTT/MSS$ which, from classical protocol theory, is a performance-based estimate of the average window size[6].

We plot p on the X axis, with the loss rate increasing to the right.

To provide a common reference for comparing data between experiments, we plot the line corresponding to the model with $C = 1$ in Figure 4 and subsequent figures.

When $p < 0.01$ (the left side of the graph) the fit between the model and the simulation data is quite plausible. By looking at the data at $p = 0.0001$ we estimate C to be 0.9, which agrees with the Delayed ACK entries in Table 1. Notice that the simulation and model have slightly different slopes, which we will investigate in Section 3.5.

When the average loss rate (p) is large (the right side of the graph), the loss of multiple packets per RTT becomes likely. If too many packets are lost, TCP will lose its Self-clock and be forced to rely on a retransmission timeout, followed by a Slow-start to recover. As mentioned above, timeouts are known not to fit the model. Note that the open markers indicate if there were *any* timeouts in the simulation for a given data point. Many of the open markers near $p = 0.01$ experienced only a few timeouts, such that the dominant behavior was still Congestion Avoidance, and the model more or less fits. By the time the loss rate gets to $p = 0.1$ the timeout behavior becomes significant. Our choice of FACK-RH TCP alters the transition between these behaviors. We compare different TCP implementations in Section 3.4.

Note that $C/\sqrt{p}$ can be viewed as the model's estimate of window size. This makes sense because packet losses and acknowledgment arrivals drive window adjustments. Although time scale and packet size do determine the total bandwidth, they only indirectly affect the window through congestion signals.

[6] In this paper, "window" always means "window in packets", and not "window in bytes."

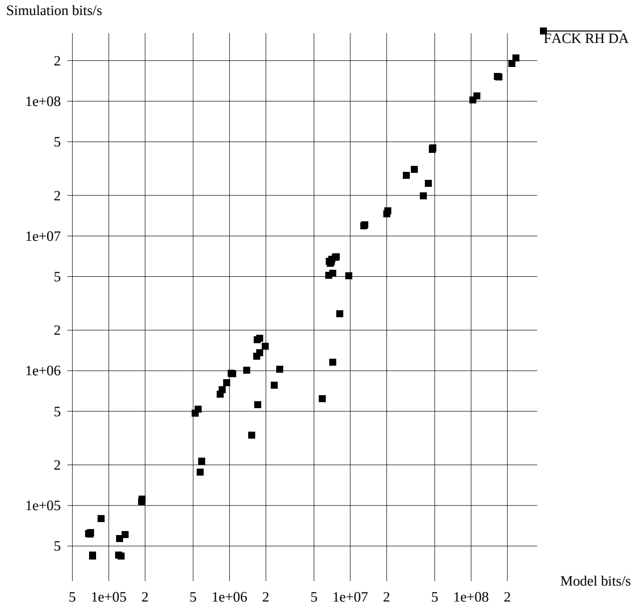

Figure 3: The Measured vs. Estimated BW
The simulation used three typical values for MSS: 536, 1460, and 4312 bytes. The one-way delay spanned from 3 ms to 300 ms; and the probability of packet loss was randomly selected between 0.00003 to 0.3. In the model we have assumed C to be 1.

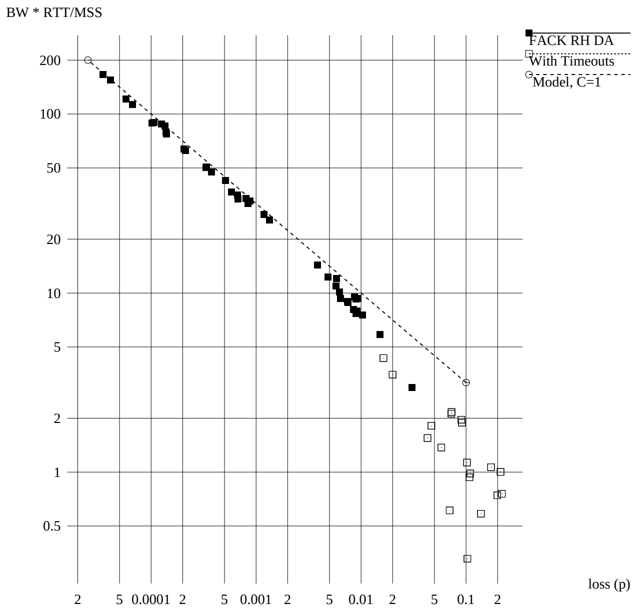

Figure 4: Window vs. Loss
This is a different view of the same data as in Figure 3. Each point has been rescaled in both axes.

3.2 Environments with Queuing

Since, under the assumptions of Section 2, the Congestion Avoidance algorithm is only sensitive to packet loss and Acknowledgement arrivals we expect the model to continue to correctly predict the window when queuing delays are experienced. Thus, with an appropriate definition for RTT, the model should hold for environments with queuing.

We performed a set of simulations using bottlenecked links where queuing could take place. We used a drop-tail link (Figure 2 with drop-tail) with $RTT = 60$ ms and $MSS = 1024$ bytes. The link bandwidth was varied from 10 kb/s to 10 Mb/s, while the queue size was varied from 5 to 30 packets. Therefore, the ratio of delay-bandwidth product to queue length spanned from 15:1 to 1:400. The simulations in Figures 5 and 6 were all performed with the stock Reno module in the simulator.

In Figure 5 we plot the data using the fixed part of the RTT, which includes only propagation delay and copy time. Clearly the fit is poor.

In Figure 6 we re-plot the same data, but use the RTT as measured by a MIB-like instrument in the simulated TCP itself. The instrument uses the round trip time as measured by the RTTM algorithm [JBB92] to compute the round trip time averaged across the entire connection. This is the average RTT as sampled by the connection itself.

This transformation has the effect of making the Y axis a measurement-based estimate of the average window. It moves individual points up (relative to the upper graph) to reflect the queuing delay.

The slope of the data does not quite agree with the model, and there are four clusters of outliers. The slope (which we will investigate in Section 3.5) is the same as in Figure 4.

The four clusters of outliers are due to the long packet times at the bottleneck link causing the Delayed ACK timer to expire. This effectively inhibits the Delayed ACK algorithm such that every data packet causes an ACK, raising C by a factor of $\sqrt{2}$ for the affected points, which lie on a line parallel to the rest of the data.

We conclude that it is necessary to use an RTT measurement that is appropriate for the connection. The RTT as sampled by the connection itself is always appropriate. Under some circumstances it may be possible to use other simpler measures of RTT, such as the time average of the queue at the bottleneck.

Reno fits the model under these conditions because the idealized topology in Figure 2 drops exactly one packet at the onset of congestion [7], and Reno's Fast

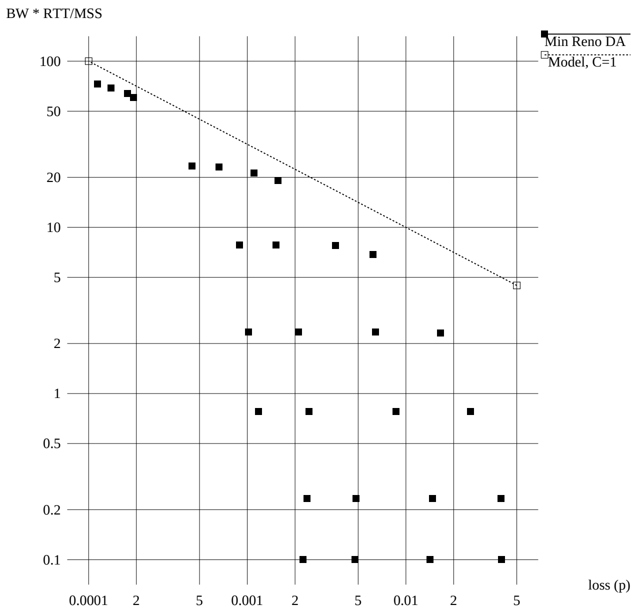

Figure 5: Estimated Window vs. Loss.
Simulations of Reno over a bottlenecked link with a drop-tail queue, without correcting for queuing delay. The RTT was 60 ms and the MSS was 1 kbyte. The bandwidth was varied from 10 kb/s to 10 Mb/s, and the queue size was varied from 5 to 30 packets.

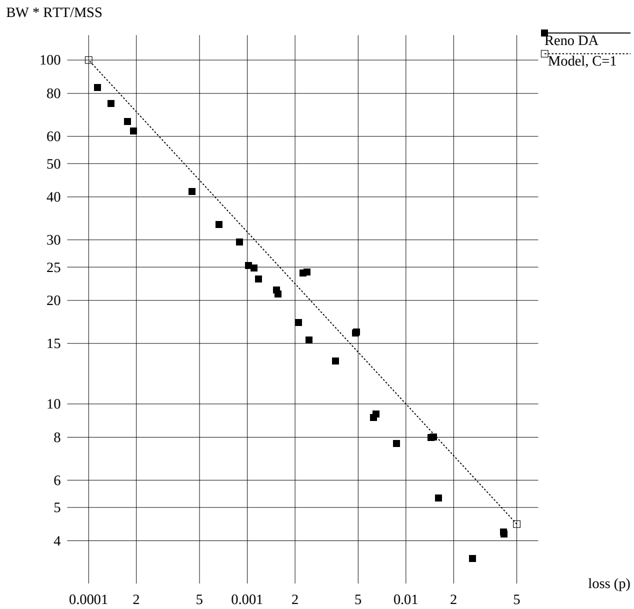

Figure 6: Estimated Window vs. Loss.
This is a different view of the same data as Figure 5, transformed by using TCP's measure of the RTT.

[7] It has been observed that Reno TCP's Self-clock is fragile in the presence of multiple lost packets within one round trip [Hoe95, Flo95, Hoe96, FF96, MM96a, LM94]. In the simulator, a single TCP connection in ongoing Congestion Avoidance nearly always causes the queue at the bottleneck to drop exactly exactly one packet when it fills. This is because the TCP opens the window very gradually, and there is no cross traffic or ACK compression to introduce jitter. Under these conditions Reno avoids any of its problems with closely spaced losses.

Recovery and Fast Retransmit algorithms are sufficient to preserve the Self-clock. Under these conditions Reno exhibits idealized Congestion Avoidance and fits the model. If the simulations are re-run using other TCP implementations with standard Congestion Avoidance algorithms [8] the resulting data is nearly identical to Figures 5 and 6. For NewReno, SACK and FACK the data points agree within the quantization errors present in the simulation instruments. This is expected, because all are either derived from the original Reno code, or were expressly designed to have the same overall behavior as Reno when subjected to isolated losses.

3.3 Phase Effects

Phase effects [FJ92] are phenomena in which a small change in path delay (on the order of a few packet times) has a profound effect on the observed TCP performance. It arises because packets leaving the bottlenecked link induce correlation between the packet arrival and the freeing of queue space at the same bottleneck. In this section we will show why phase effects are consistent with the model, and what this implies about the future of Internet performance instrumentation.

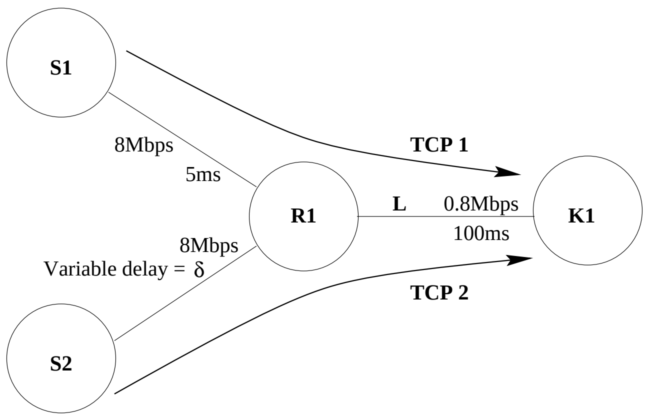

Figure 7: Phase Effects topology.

[8] The simulator includes models for several different TCP implementations. Tahoe [Jac88a] and Reno (described in [Ste97] and [Jac90]) are well known. The simulator also includes a SACK implementation "SACK1" [Flo96], which was based on the original [JB88] SACK, but has been updated to RFC 2018 [MMFR96]. This is, by design, a fairly straightforward implementation of SACK TCP using Reno-style congestion control. NewReno is a version of Reno that has some modifications to correct what is essentially a bug that frequently causes needless timeouts in response to multiple-packet congestion signals. This modification was first suggested by Janie Hoe [Hoe95, CH95] and has been thoroughly analyzed [Flo95, FF96].

Tahoe TCP has significantly different steady-state behavior than newer TCP implementations. Whenever a loss is detected the congestion window is reduced to 1 (without changing *ssthresh*). This causes a Slow-start, taking roughly $log_2 W$ round trips, and delivering roughly W segments). Tahoe does not fit the model in a badly underbuffered network (due to persistent repeated timeouts). At higher loss rates when a larger fraction of the overall time is spent in Slow-Start, Tahoe has a slightly different shape, and therefore the model is less accurate.

In Figure 7 we have reconstructed[9] one of the simulations from [FJ92], using two SACK TCP connections through a single bottlenecked link with a drop-tail router. Rather than reconstructing the complete simulation in which the variable delay is adjusted across many closely spaced values, we present a detailed analysis of one operating point, $\delta = 9.9$ ms. In this case, the packets from connection 2 are most likely to arrive just after queue space has been freed, but just before packets from connection 1. Since the packets from connection 2 have a significant advantage when competing for the last packet slot in the queue, connection 1 sees more packet drops.

The packets are 1 kbyte long, so they arrive at the receiver (node $K1$) every 10 ms. The 15 packet queue at link L slightly underbuffers the network. We added instrumentation to both the bottlenecked link and the TCP implementations, shown in Table 2. The **Link** column presents the link instruments on L, including total link bandwidth and the time average of the queue length, expressed as the average queuing delay. The two **TCP** columns present our MIB-like TCP instruments for the two TCP connections, except for the **RTT Estimate** row, which is the average RTT computed by adding the average queue length of the link to the minimum RTT of the entire path.

The loss instruments in the TCP implementation categorize each loss depending on how it affected the congestion window. Losses that trigger successful (clock-preserving) divide-by-two window adjustments are counted as "CA events". All other downward window adjustments (i.e. timeouts) are counted as "non-CA events". Additional losses which are detected while already in recovery and do not cause their own window adjustments and are counted as "other losses"[10]. In the drop-tail case (on the left side of the table), we can see that TCP1 experienced 103 CA events and 37 non-CA events (timeouts). During those same recovery intervals, there were an additional 76 losses which were not counted as congestion signals. Note that p is the number of CA events per acknowledged segment. The link loss instruments, by contrast, do not categorize lost packets, and cannot distinguish losses triggering congestion avoidance.

The **TCP RTT** instrument is the same as in the previous section (i.e. based on the RTTM algorithm).

Note that even though the delay is different by only 4.9 ms there is about a factor of 4 difference in the performance. This is because the loss rate experienced by

[9] Our simulation is identical, except that we raised the receiver's window such that it does not interfere with the Congestion Avoidance algorithm. This alters the overall behavior somewhat because the dominant connection can potentially capture the full bandwidth of the link.

[10] Every loss episode counts as exactly one CA or non-CA event. Episodes in which there was a Fast Retransmit, but Fast Recovery was unsuccessful at preserving the Self-clock or additional losses caused additional window reductions were counted as non-CA events.

All additional retransmissions (occurring in association with either a timeout or congestion signal) are counted as additional lost packets.

Table 2: Phase effects with queue limit = 15, $\delta = 9.9$

Drop Tail			$\delta = 9.9ms$		RED		
Link	TCP1	TCP2			Link	TCP1	TCP2
781	133	648	Bandwidth	kb/s	798	430	368
259	103+37+76	41+0+2	losses (CA+timo+other)		139	64+0+1	73+0+1
48795	8287	40508	packets		49853	26851	23002
0.0053	0.0124	0.0010	p		0.0028	0.0024	0.0032
	325	315	TCP RTT	ms		304	307
83.49			Link Delay	ms	77.47		
	305	315	RTT Estimate	ms		299	309
	288 (118%)	279 (57%)	Link Model	kb/s		405 (6%)	393 (7%)
	177 (34%)	638 (2%)	TCP Model	kb/s		432 (1%)	370 (1%)

Table 3: Phase effects with queue limit = 100, $\delta = 9.9$

Drop Tail			$\delta = 9.9ms$		RED		
Link	TCP1	TCP2			Link	TCP1	TCP2
800	244	556	Bandwidth	kb/s	798	401	397
20	12+0+3	5+0+0	losses (CA+timo+other)		137	68+0+0	69+0+0
50000	15250	34750	packets		49845	25039	24806
0.0004	0.0008	0.0001	p		0.0027	0.0027	0.0028
	1029	1029	TCP RTT	ms		304	307
801.03			Link Delay	ms	77.25		
	1022	1032	RTT Estimate	ms		299	308
	313 (29%)	310 (45%)	Link Model	kb/s		409 (3%)	396 (1%)
	222 (10%)	518 (7%)	TCP Model	kb/s		404 (1%)	395 (1%)

each connection is different by an order of magnitude.

The model is used to predict performance in two different ways. The first technique, the **Link Model**, uses only the link instruments, while the second, the **TCP Model**, uses only the TCP instruments. Clearly applying the model to the aggregate link statistics or to TCP1 statistics (with 37 timeouts) in the drop-tail case can not yield accurate results. The model[11] applied to TCP2's internal instruments correctly predicts the bandwidth.

Random Early Detection (RED) [FJ93] is a form of Active Queue Management [B+97], which manages the queue length in a router by strategically discarding packets before queues actually fill. Among many gains, this permits the router to randomize the packet losses across all connections, because it can choose to drop packets independent of the instantaneous queue length, and before it is compelled to drop packets by buffer exhaustion.

In the phase effects paper [FJ92], it is observed that if a router uses RED instead of drop-tail queuing, the phase effects disappear. In the right side of Table 2 we present a simulation which is identical to the left side, but using RED at the bottleneck (link L). With RED, the link instruments are in nearer agreement with the TCP instruments; so the model gives fairly consistent results when calculated from either link statistics or TCP instruments[12]. The residual differences between the results predicted by the model are due to p not being precisely uniform between the two TCP connections. This may reflect some residual bias in RED, and bears further investigation.

In Table 3 we repeated the simulations from Table 2, but increased the packet queue limit at link L to 100. As you would expect, this only slightly changes the RED case. However, there are several interesting changes to the drop tail case.

Average RTT has risen to a full second. Without RED to regulate the queue length, SACK TCP only halves its window when it fills the 100 packet queue. TCP's window is being regulated against a full queue, rather than some other operating point closer to the onset of queued data. Even if both connections experience a packet loss in the same RTT, the queue will not fully drain. We can gauge the queue sizes from the average link delay instrument: 800 ms corresponds to 80 packets. We know that the peak is 100 packets, so the minimum queue is likely to be near 60 packets, or 600 ms! This is not likely to please interactive users.

The tripling of the RTT requires an order of magnitude lower loss to sustain (roughly) constant bandwidth. The model is less accurate, even when applied to the TCP instruments because the loss sample size is too small, causing a large uncertainty in p. Excluding the initial Slow-start, TCP2 only experienced 5 losses

[11] Since we are most interested in the drop-tail simulation near $p = 0.01$, we estimated a locally-accurate value of $C = 0.8$ by examining the data used in Figure 6 in the previous section. This value of C was used for all the model bandwidths shown in Tables 2 and 3.

[12] Note that RED also lowered the average link delay, lowered the total packet losses, raised the aggregate throughput. RED usually signals congestion with isolated losses. Therefore Reno might operate as well as SACK TCP in this environment.

during the 500 second measurement interval (i.e. each Congestion Avoidance cycle took 100 seconds!)

The symptoms of overbuffering without RED are: long queuing delays and very long convergence time for the congestion control algorithm.

Also note that our opening problem of projecting end-to-end TCP performance from hop-by-hop path properties requires reasonable assurance that the link statistics collected at any one hop are indicative of that hop's contribution to the end-to-end path statistics. This requirement is not met with drop-tail routers, where correlation in the traffic causes correlation in the drops.

If packet losses are not randomized at each bottleneck, then hop-by-hop performance metrics may not have any bearing upon end-to-end performance. RED (or possibly some other form of Active Queue Management) is required for estimating end-to-end performance from link statistics. Conversely, if a provider wishes to assure end-to-end path performance, then all routers (and other potential bottlenecks) must randomize their losses across all connections common to a given queue or bottleneck.

Also note that if the losses are randomized, $C/\sqrt{p}$ is a bound on the window size for all connections through any bottleneck or sequence of bottlenecks. Furthermore, connections which share the same (randomized loss) bottleneck *tend to equalize* their windows [CJ89]. We suspect that this is the implicit resource allocation principle already in effect in the Internet today.[13]

3.4 Effect of TCP Implementation

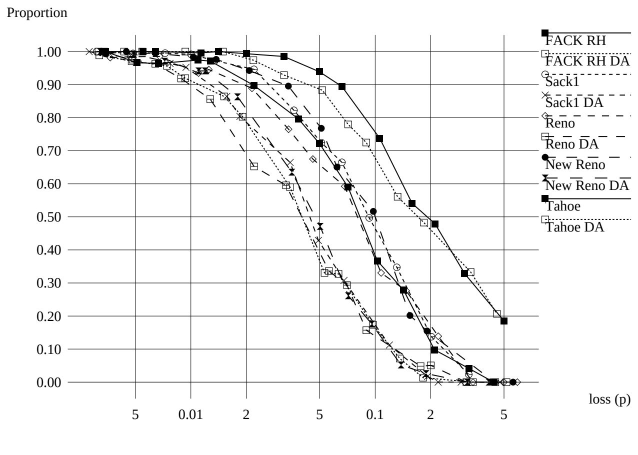

Figure 8: Algorithm dominance vs packet loss. The fraction of all downward window adjustments which are successful (clock-preserving) divide-by-two window adjustments.

[13] Note that our observation is independent of the model in this paper. To the extent that the window is only determined by losses (which isn't quite true) and that losses are equalized at bottlenecks (which also isn't true without RED, etc.), the Internet *must* tend to equalize windows.

We wish to compare how well different TCP implementations fit the model by investigating two aspects of their behavior. We first investigate the transition from Congestion Avoidance behavior at moderate p to timeout driven behavior at larger p. In the next section, we investigate a least squares fit to the model itself.

As mentioned earlier, the model does not predict the performance when TCP is timeout driven. Although in our simulations timeouts do not cause a serious performance penalty, we have not included cross traffic or other effects that might raise the variance of the RTT and thus raise the calculated retransmission timeout. Although Figure 4 might seem to imply that the model fits timeouts, remember that this was in a queueless environment, where there is zero variance in the RTT. Under more realistic conditions it is likely that timeouts will significantly reduce the performance relative to the model's prediction.

We simulated all of the TCP implementations supported by the simulator[8], with and without Delayed ACK receivers, and instrumented the simulator to tabulate all downward window adjustments into the same two categories as used in the previous section. The first, "CA events," includes all successful (clock preserving) divide-by-two window adjustments. The second, "non-CA events", includes all other downward window adjustments. In Figure 8 we plot the proportion of all downward adjustments which were successful invocations of the Congestion Avoidance algorithm. (This data is also summarized on the right side of Table 4).

FACK-RH TCP avoids timeouts under more severe loss than the other TCP implementations because it normally sends one segment of new data after the first duplicate acknowledgment but before reaching the *dupack* threshold (triggering the first retransmission– see Appendix A). All of the other TCP's are unable to recover from a single loss unless the window is at least 5 packets[14]. The horizontal position of the steep downward transition reflects the loss rate at which the various TCPs no longer retain sufficient average window for Fast Retransmit. Under random loss SACK, Reno, NewReno, and Tahoe all have essentially the same characteristics.

3.5 Fitting the slope

As we have observed in the previous sections, the window vs. loss data falls on a fairly straight line on a log-log plot, but the slope is not quite $-1/2$. This suggests that a better model might be in the following form:

$$BW = \frac{MSS}{RTT} C p^k \qquad (5)$$

Where k is roughly $-1/2$.

We performed a least mean squared fit between Equation 5 and the TCP performance as measured in the simulator. The results are shown in Table 4. All

[14] One packet is lost, the next three cause duplicate acknowledgements, which are only counted. The lost packet is not retransmitted until the fifth packet is acknowledged.

Acknowledgement Scheme	TCP Implement-ation	Least Mean Squares fit				Proportion of successful $W/2$ adjustments		
			Equation 3	Equation 5				
		N	C	k	C	$p = .01$	$p = 0.033$	$p = 0.1$
No Delayed ACKs	FACK	16	1.352 ± 0.090	-0.513	1.205 ± 0.047	0.996	0.985	0.738
	SACK	11	1.346 ± 0.052	-0.508	1.247 ± 0.033	0.992	0.822	0.497
	Reno	12	1.331 ± 0.054	-0.521	1.096 ± 0.009	0.935	0.765	0.331
	New Reno	12	1.357 ± 0.055	-0.516	1.167 ± 0.020	0.983	0.896	0.517
	Tahoe	11	1.254 ± 0.079	-0.534	0.920 ± 0.015	0.974	0.796	0.367
Delayed ACKs	FACK DA	15	0.928 ± 0.086	-0.519	0.783 ± 0.045	1.000	0.929	0.725
	SACK DA	10	0.938 ± 0.036	-0.518	0.792 ± 0.012	0.952	0.664	0.112
	Reno DA	10	0.939 ± 0.046	-0.524	0.752 ± 0.015	0.919	0.595	0.157
	New Reno DA	11	0.935 ± 0.045	-0.526	0.738 ± 0.006	0.942	0.635	0.176
	Tahoe DA	11	0.883 ± 0.076	-0.542	0.596 ± 0.012	0.919	0.590	0.173

Table 4: Comparison of various TCP implementations.

simulations which experienced timeouts were excluded, so the fit was applied to runs exhibiting only the Congestion Avoidance algorithm[15]. The number of such runs are shown in column N.

For $k = -0.5$, the values of C are quite close to the derived values. The quality of the fit is also quite good. As expected, Delayed Acknowledgements change C by $\sqrt{2}$. When k is allowed to vary slightly, the fit becomes even better still, and the best values for k are only slightly off from -0.5. This slight correction to k probably reflects some of the simplifying assumptions used in the derivation of Equation 3. One simplification is that TCP implementations perform rounding down to integral values in several calculations which update *cwnd*. The derivation of the model assumes *cwnd* varies smoothly, which overestimates the total amount of data transferred in a congestion avoidance cycle. Another simplification is that the model expects the window to begin increasing again immediately after it is cut in half. However, recovery takes a full RTT, during which TCP may not open the window. We plan to investigate the effects of these simplifications in the future.

4 TReno results

Much of our experimentation in TCP congestion dynamics has been done using the TReno performance diagnostic [Mat96]. It was developed as part of our research into Internet performance measurement under the IETF IP Performance Metrics working group [Mat97]. TReno is a natural succession to the windowed ping diagnostic [Mat94b]. (The FACK-RH algorithm for TCP is the result of the evolution of the congestion control implemented in TReno.)

TReno is designed to measure the single stream bulk transfer capacity over an Internet path by implementing TCP Congestion Avoidance in a user mode diagnostic tool. It is an amalgam of two existing algorithms: traceroute [Jac88b] and an idealized version of TCP congestion control. TReno probes the network with ei-

[15] FACK fits less well because it avoids timeouts and thus includes data at larger p, where rounding terms become significant.

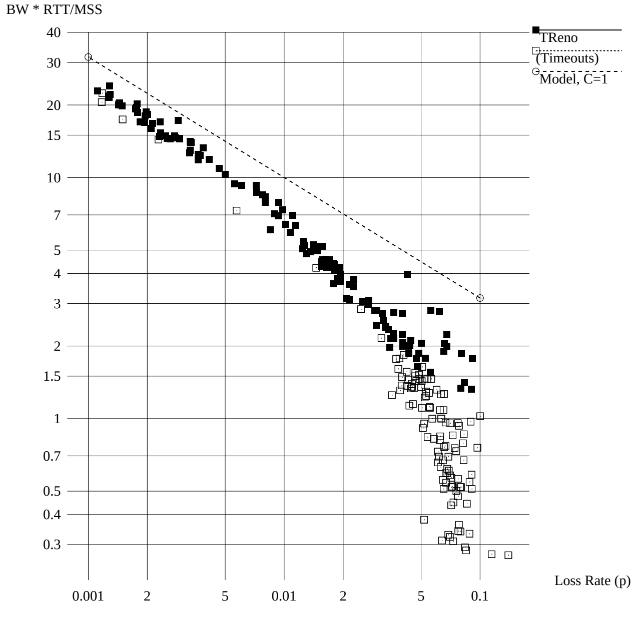

Figure 9: TReno measurement data
This data fits Equation 3 with $C = 0.744 \pm 0.193$ or to Equation 5 with $k = -0.617, C = 0.375 \pm 0.054$. These are poorer than the fits in Table 4, in part because the TReno data extends to much worse loss rates, where the effects of rounding become more pronounced. The C values are lower, indicating about 20% lower performance than TCP in a simulator.

ther ICMP ECHO packets (as in the ping program), or low-TTL UDP packets, which solicit ICMP errors (as in the traceroute program). The probe packets are subject to queuing, delay and congestion-related loss comparable to TCP data and acknowledgment packets. The packets carry sequence numbers which are reflected in the replies, such that TReno can always determine which probe packet caused each response, and can use this information to emulate TCP.

This has an advantage over true TCP for experimenting with congestion control algorithms because TReno only implements those algorithms and does not need to implement the rest of the TCP protocol, such as the three-way SYN handshake or reliable data delivery. Furthermore, TReno is far better instrumented than any of today's TCP implementations. Thus it is a good vehicle to test congestion control algorithms over real Internet paths, which are often not well-represented by the idealized queuing models used in simulations.

However, TReno has some intrinsic differences from real TCP. For one thing, TReno does not keep any state (corresponding to the TCP receiver's state) at the far end of the path. Both the sender's and receiver's behaviors are emulated at the near end of the path. Thus it has no way to distinguish between properties (such as losses or delay) of the forward and reverse paths[16].

For our investigation, TReno was run at random times over the course of a week to two hosts utilizing different Internet providers. Due to normal fluctuation in Internet load we observed nearly two orders of magnitude fluctuations in loss rates. Each test lasted 60 seconds and measured model parameters p and RTT from MIB-like instruments.

The TReno data[17] is very similar to the simulator data in Figure 4, except that the timeouts have a more profound negative impact on performance. If the runs containing timeouts are neglected, the data is quite similar.

Also note that TReno suffered far more timeouts than FACK-RH in the simulator, even though they have nearly identical internal algorithms. This is discussed in the next section, where we make similar observations about the TCP data.

5 TCP measurements

In this section, we measured actual TCP transfers to two Internet sites in order to test the model. This was done by using a slightly modified version of "tcptrace" [Ost96] to post-process packet traces to reconstruct p and RTT from TCP's perspective. These instruments are nearly identical to the instruments used in the TCP simulations.

[16] If the ICMP replies originate from a router, (such as intermediate traceroute hops) TReno may suffer from a low performance ICMP implementation on the router. This is not an issue in the data presented here, because the end systems can send ICMP replies at full rate.

[17] TReno emulates Delayed Acknowledgements.

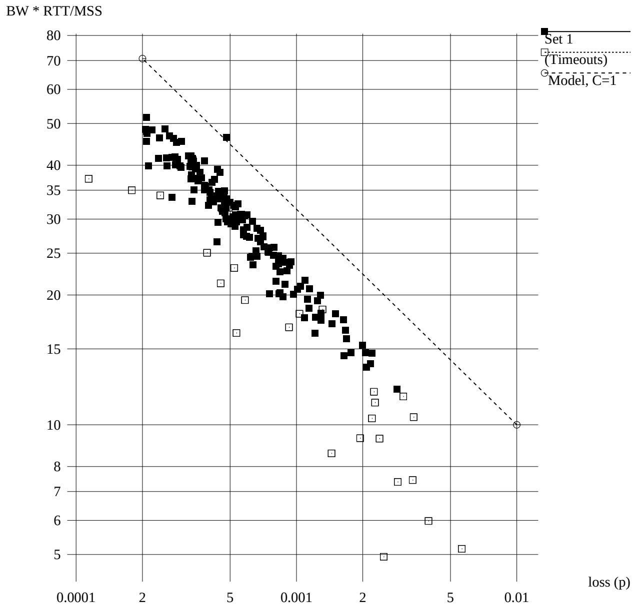

Figure 10: Measured TCP data, Set 1
This data fits Equation 3 with $C = 0.700 \pm 0.057$ or to Equation 5 with $k = -0.525, C = 0.574 \pm 0.045$. These values for C are about 25% lower than TCP in a simulator.

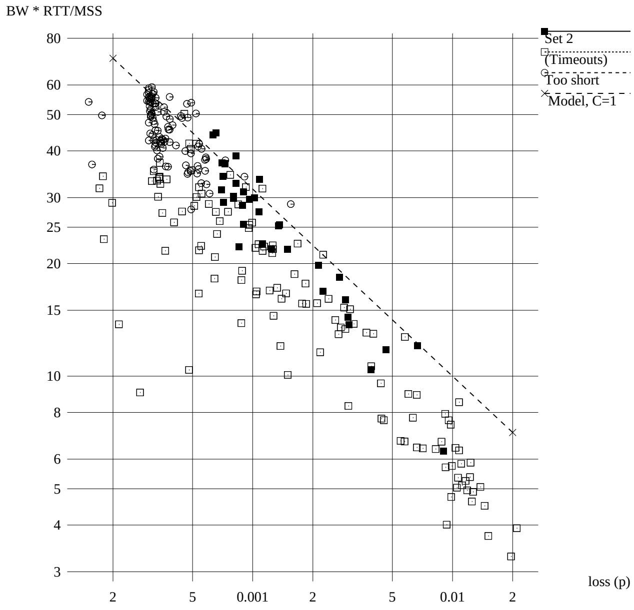

Figure 11: Measured TCP data, Set 2
This data fits Equation 3 with $C = 0.908 \pm 0.134$ or to Equation 5 with $k = -0.611, C = 0.418 \pm 0.058$. Some of the individual data points are above the $C = 1$ reference line. The live Internet measurements were not over long enough intervals to permit trimming the Slow-start overshoot described in Appendix B. This data set may have been subject to Slow-start overshoot.

This experiment proved to be far more difficult than expected. We encountered a number of difficulties with the test paths themselves. The Internet is vastly noisier and less uniform than any simulation [Pax97a, Pax97b]. Furthermore, several paths exhibited behaviors that are beyond the scope of the model[18].

The tests were run at random times over the course of 10 days, at an average rate of once per hour to each remote site. The period included a holiday weekend (with unusually low background traffic), and was not the same week as the TReno data.

During our testing, the connections transferred as much data as possible in 100 seconds of elapsed time to two different Internet sites. Figure 10 shows that the model fits fairly well to data collected to one target. If you compare this to Figure 4 it is in reasonable agreement, considering the difference in scale.

Figure 11, on the other hand, does not fit as well. It is illustrative to dissect the data to understand what is happening over this path, and how it relates to the model's applicability. Our first observation is that there are too many timeouts (indicated by open circles), considering the low overall loss rate.

To diagnose this phenomenon, we looked at the raw packet traces from several of the transfers. Nearly all of the timeouts were the result of losing an entire window of consecutive packets. These short "outages" were not preceded by any unusual fluctuation in delay. Furthermore, the following (Tahoe-style) recovery exhibited no SACK blocks or step advances in the acknowledgment number. Therefore an entire window of data had been lost on the forward path. This phenomenon has been observed over many paths in the Internet [Pax97b, p305] and is present in Figures 9 and 10, as well. Lore in the provider community attributes this phenomenon to an interaction between routing cache updates and the packet forwarding microcode in some commercial routers[19].

Our second observation (regarding traces without timeouts) is that the number of "CA events" is very small, with many traces showing 3 or fewer successful window halving episodes. An investigation of the trace statistics reveals that the path had a huge maximum round trip time (1800 ms), and that during some of our test transfers the average round trip time was as large as a full second. This suggests that the path is overbuffered and there is no active queue management in effect to regulate the queue length at the bottleneck.

Real TCP over this path exhibits the same symptoms as the simulation of an overbuffered link without RED in Section 3.3: long queuing delays and very long cycle times for the congestion avoidance algorithm. As a consequence, our 100 second measurement interval was not really long enough and captured only a few congestion signals, resulting in a large uncertainty in p. The open circles on the left side of Figure 11 have observable vertical banding in the data corresponding to 1, 2 or 3 total congestion avoidance cycles.[20]. Traces with 4 or more congestion avoidance cycles are included in the good data (solid squares).

Our test script also used conventional diagnostic tools to measure background path properties bracketing the TCP tests. Although we measured several parameters, the RTT statistics were particularly interesting. For "not under test" conditions, the minimum RTT was 72.9 ms, and the average RTT was 82 ms[21]. From the tcptrace statistics, we know that during the test transfers the average RTT rose to 461 ms[22].

Our TCP transfers were sufficient to substantially alter the delay statistics of the path. We believe this to be an intrinsic property of Congestion Avoidance: any long-running TCP connection which remains in Congestion Avoidance raises the link delay and/or loss rate to find its share of the bottleneck bandwidth. Then Equation 3 will agree with the actual bandwidth for the connection, and if the losses at the bottleneck are sufficiently randomized, the link statistics (delay and loss) will be common to all traffic sharing the bottleneck.

In general, the current Internet does not seem to exhibit this property. We suspect that this is due to a combination of effects, including Reno's inability to sustain TCP's Self-clock in the presence of closely-spaced losses, the prevalence of drop-tail queues and faulty router implementations.

6 Multiple Congested Gateways

In this section we apply the model to the problem of TCP fairness in networks with multiple congested gateways. Floyd published a simulation and heuristic analysis of this problem in 1991 [Flo91]. In this paper, she analyzed the following problem: given a network of $2n$ gateways, where n are congested by connections that use only one of the congested gateways, what portion of the available bandwidth will a connection passing through all n congested gateways receive? The analysis of this problem presented by Floyd computes bandwidth by determining the packet loss rate for each connection[23]. Here, we demonstrate that the same re-

[18] One discarded path suffered from packet reordering which was severe enough where the majority of the retransmissions were spurious.

[19] Note that burst losses and massive reordering are not detectable using tools with low sampling rates. These sorts of problem can most easily be diagnosed in the production Internet with tools that operate at normal TCP transfer rates.

[20] The bands are at roughly $p = 0.00015, 0.00035$ and 0.0005.

[21] Each background measurement consisted of 200 RTT samples taken either shortly before or shortly after each test TCP transfer. The median of the measurement averages was 80 ms for the "not under test" case.

[22] The median of the measurement averages was 466 ms for the "under test" cases.

Unfortunately, the burst losses obscured our background loss rate measurement, because in the average they caused *much more packet loss than the true congestion signals*. Since they caused TCP timeouts, they were implicitly excluded from the TCP data, but not from our background measurements.

Note that to some extent the burst losses and the RED-less overbuffering are complementary bugs because each at least partially mitigates the effects of the other.

[23] We should note that Floyd's work was published three years prior to Mathis [Mat94a] and five years prior to Ott [OKM96a].

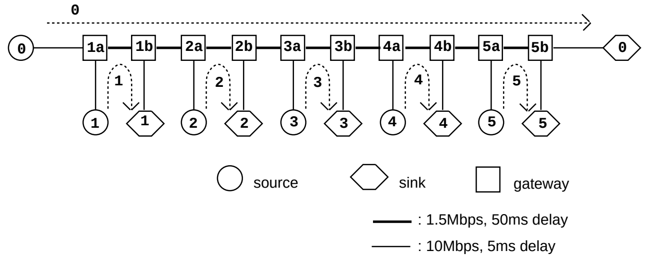

Figure 12: The multiple congested gateways scenario.

sults can be obtained by using Equation 3.

Figure 12 (from [Flo91]) shows the exact scenario we are interested in analyzing. Each dotted line indicates a connection from a source to a sink. Generalizing the parameters, we define δ to be the individual delay of the long links (50 ms in Figure 12) and ϵ to be the delay of the short links (5 ms in Figure 12). The round-trip delay for Connection i is:

$$\delta_i = 2\delta + \delta_Q + 4\epsilon \tag{6}$$

Here, we add a term δ_Q which represents the average delay due to queuing at the bottleneck router.[24] The round-trip delay for Connection 0 is:

$$\delta_0 = 2(2n-1)\delta + n\delta_Q + 4\epsilon \tag{7}$$

The model can then be used to predict the bandwidth for each connection:

$$B_i = C\frac{MSS}{\delta_i}\frac{1}{\sqrt{p}} \tag{8}$$

$$B_0 = C\frac{MSS}{\delta_0}\frac{1}{\sqrt{np}} \tag{9}$$

The total link bandwidth used at each congested hop is given by the sum of these two values:

$$B = B_0 + B_i = C\frac{MSS}{\sqrt{p}}\left(\frac{1}{\sqrt{n}\delta_0} + \frac{1}{\delta_i}\right) \tag{10}$$

The fraction of the bandwidth used by Connection 0 is then given by (divide Equation 9 by Equation 10):

$$\frac{B_0}{B} = \frac{1}{1+\frac{\delta_0\sqrt{n}}{\delta_i}} = \frac{1}{1+\sqrt{n}\left(\frac{2(2n-1)\delta+n\delta_Q+4\epsilon}{2\delta+\delta_Q+4\epsilon}\right)} \tag{11}$$

The first part of Equation 11 exactly matches Floyd's result [Flo91, Claim 5, Equation 3][25]. The second part of Equation 11 fills in the precise formulae for the delays. If we assume δ_Q and ϵ are small, we again match Floyd's results [Flo91, Corollary 6].

[24] We make the assumption that δ_Q is the same for all of the connections being considered. This is probably not a realistic assumption, but it does allow us to simplify the problem somewhat.

[25] Noting that we are using an increase-by-1 window increase algorithm, which is identical for both the long and short connections.

$$\frac{B_0}{B} \approx \frac{1}{1+\sqrt{n}(2n-1)} \tag{12}$$

In the case of overbuffered, drop-tail gateways, where δ_Q is large and phase effects are not an issue, we get a slightly different result:

$$\frac{B_0}{B} \approx \frac{1}{1+n^{3/2}} \tag{13}$$

It is useful to note that C has dropped out of the calculation. A precise estimate of C was not needed.

In this section, we have used Equation 3 to estimate TCP performance and bandwidth allocation in a complex network topology. Our calculation agrees with the prior heuristic estimates for this environment.

7 Implications for the Internet

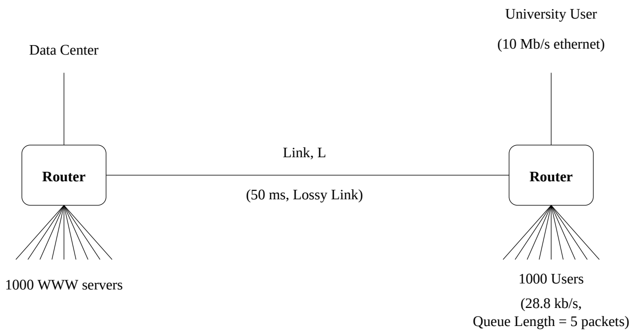

Figure 13: Simplified Internet Topology

In this section we use Equation 3 to investigate some properties of traffic and congestion in the global Internet. Figure 13 is a very simplified schematic of the Internet. To the left are information providers, including a large population of WWW servers and other large data centers, such as supercomputing resources.

To the right are information consumers, including a large population of modem-based users and a smaller population of Research and Education users, who are trying to retrieve data from the data center.

The link between the data suppliers and consumers represents a long path through an intra-continental Internet. For illustration purposes we assume that the path can be modeled as having a single bottleneck, such that the entire path has a fixed average delay and a variable loss rate due to the total load at the bottleneck. Furthermore we assume that each individual data supplier or consumer shown in Figure 13 presents an insignificant fraction of the total load on the link. The loss rate on L is determined by the aggregate load (of *many* modems and R&E users) on the link and is uniform across all users. Thus the individual connections have no detectable effect upon the loss rate.

We wish to investigate how changes in the loss rate at link L might affect the various information consumers and to explore some strategies that might be used to

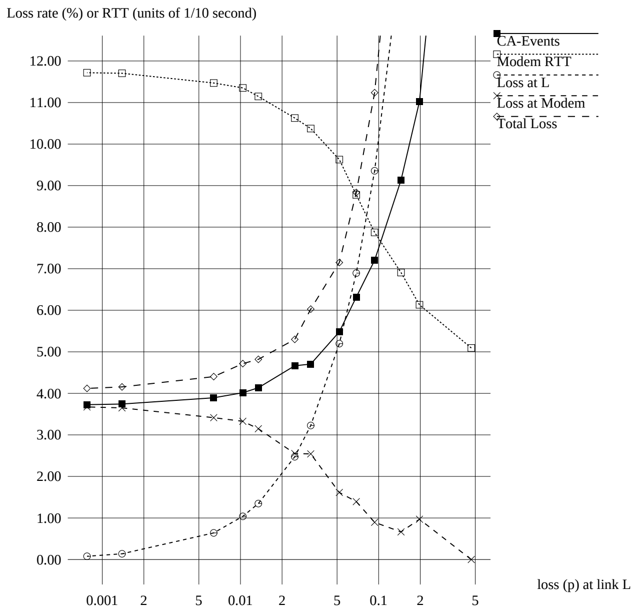

Figure 14: Packet loss experienced by modem users. Packet loss on link L is largely offset by reduced loss at the modem. "Total loss" is computed from the link instruments, "CA-Events" from the instruments in the TCP. (This is for FACK-RH TCP).

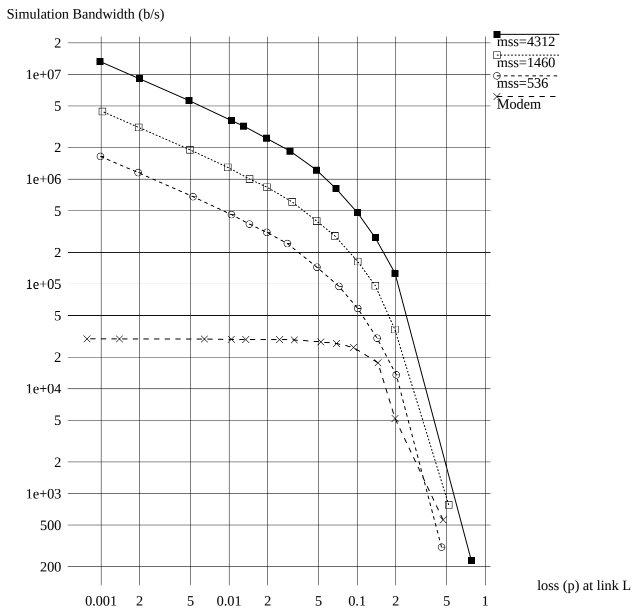

Figure 15: Effect of loss rate at link L. While modem users (with a 1000 byte MSS) do not notice loss on link L until it is quite high, R&E users suffer severe performance degradation, particularly when using a small MSS.

control the elapsed time needed to move scientific data sets.

First we consider TCP congestion control at the modems, which are the likely bottlenecks for the low end users. A full-sized packet (1000 bytes) takes about 277 ms of modem transmission time. The RTT of the unloaded path is about 400 ms (1000 byte packets flowing in one direction, 40 byte acknowledgements flowing in the other). Assume 5 packets of queue space at the modem, then the maximum window size is between 6 and 7 packets, so the "half window" must be roughly 3 packets, yielding an average window of about 5 packets. Since the data packets arrive at the clients with more than 200 ms headway, acknowledgements will be sent for every segment (due to the Delayed Acknowledgement timer). Thus we assume that $C = 1.2$ (the non-Delayed Acknowledgement periodic case from Table 1), so the 5 packet average window requires a loss rate of roughly 4% ($p = 0.04$). This can be observed on the far left edge of Figure 14. (We are assuming FACK-RH TCP).

The packet loss at the modem provides feedback to the TCP sender to regulate the queue at the modem. This queue assures that the modem utilization is high, such that data is delivered to the receiver every 277 ms. These data packets cause the receiver to generate acknowledgements every 277 ms, which clock more data out of the server at a nearly constant average rate.

Next we want to consider what happens if link L loses about 3% of the packets. Since this is not enough to throttle TCP down to 28.8 kb/s, the modem must still be introducing some loss, but less than before. Since the modem is still introducing loss, it must still have a significant average queue, so the server still sends data every 277 ms. With any SACK TCP (including FACK-RH), only the missing data is retransmitted, so the average goodput for the modem user continues to be 28.8 kb/s. Therefore the 3% loss on link L has an insignificant effect on the modem user.

As the loss rate rises beyond 3%, the queue at the modem becomes shorter, reducing the RTT from about 1.2 seconds down toward 400 ms. Note that a 5 packet queue overbuffers the path, so the utilization does not start to fall until the loss rate approaches 10%[26].

Now consider the plight of an R&E user (See Figure 15). What performance limitations are imposed on the R&E user by 3% loss? From Equation 3, the average window size must be about 5 packets. If these are 536 byte packets (with a 100 ms RTT), the user can get no more than about 250 kb/s. Timeouts and other difficulties could further reduce this performance. At 250 kb/s, moving 1 Gigabyte of data[27] takes over 8 hours.

[26] Note that this is FACK-RH TCP, which does substantially better than other TCPs in this region.

Many recovery episodes exhibit multiple dropped packets (note that the total link loss rate and CA-Events differ) so Reno has no hope of preserving its Self-clock. As the peak window size falls below 5 packets, conventional Fast Retransmit will also fail.

[27] Note that at today's prices, with disk space available at below $100 per Gigabyte, workstations commonly have several Gigabytes of disk space. It is not at all unusual for researchers to

As a consequence, some researchers have been known to express mail tapes instead of using the Internet to transfer data sets.

Suppose the R&E user needs to move 1 Gigabyte of data in 2 hours. This requires a sustained transfer rate of about 1 Mb/s. What loss rate does the user need to meet this requirement? Assume $C < 1$ (because the R&E receivers will be using Delayed Acknowledgments) and invert Equation 4 to get a bound on p:

$$p < \left(\frac{MSS}{BW RTT}\right)^2 \tag{14}$$

The model predicts that the R&E user needs a loss rate better than 0.18% ($p = 0.0018$) with 536 byte packets. At 1460 bytes, the maximum loss rate rises to 1.4%. If the R&E user upgrades to FDDI (and uses 4312 byte packets), Equation 14 suggests that the network only needs to have less than 11% loss.

In practice, we need to consider the actual value of C and potential bottlenecks in all other parts of the system, as well as the details of the particular TCP implementation. This calculation using the model agrees with the simulation shown in Figure 15.

Note that the specific results in this section are very sensitive to many of our assumptions, especially to the use of FACK-RH TCP and the 5 packet queue at the modem. Different assumptions will change the relative positions of the data in our graphs, but the overall trends are due to intrinsic properties of TCP congestion control and the Congestion Avoidance algorithm.

We can draw some useful rules-of-thumb from our observations. First, each factor of 3 in the MSS (4312 to 1460, or 1460 to 536) lowers the required end-to-end loss rate by nearly an order of magnitude. Furthermore, a network which is viewed as excellent by modem users can be totally inadequate for a Research and Education user.

8 Conclusion

We have shown, through simulation and live observations, that the model in Equation 3 can be used to predict the bandwidth of Congestion Avoidance-based TCP implementations under many conditions.

In the simulator all presented TCPs fit the model when losses are infrequent or isolated. However, since different TCPs vary in their susceptibility to timeouts, they diverge from the model at different points.

Live Internet tests show rough agreement with the model in cases where no pathological behaviors are present in the path.

The model is most accurate when using delay and loss instruments in the TCP itself, or when loss is randomized at the bottleneck. With non-randomized losses, such as drop-tail queues, the model may not be able to predict end-to-end performance from aggregate link statistics.

FACK-RH, which treats multiple packet losses as single congestion signals, fits the model across a very wide range of conditions. Its behavior is very close to ideal TCP Congestion Avoidance. Reno, on the other hand, stumbles very easily and deviates from the model under fairly ordinary conditions.

To produce a model that applies to *all* loss rates, we need to have a model for timeout-driven behavior.

Overbuffering without RED or some other form of queue management does not interact well with SACK TCP. A single pair of end-systems running SACK over a long Internet path without RED are likely to sustain persistent, unpleasantly long queues.

The model can be used to predict how TCP shares Internet bandwidth. It can also be used to predict the effects of TCP upon the Internet, and represents an equilibrium process between loss, delay and bandwidth.

9 Acknowledgements

We would like to thank Sally Floyd and Steve McCann (LBL's ns simulator), as well as Shawn Ostermann (tcptrace) for making their software publicly available, without which we would have been unable to complete this work. We would also like to thank Dr. Floyd for allowing us to use Figure 12 from [Flo91]. We appreciate the willingness of Mark Allman, Kevin Lahey, and Hari Balakrishnan to allow us to use equipment at their sites for our remote testing. We would also like to acknowledge Bruce Loftis, for his assistance in fitting parameters to the data, and Susan Blackman, for making suggestions to improve the readability of this paper.

A FACK-RH TCP

The FACK-RH TCP used in the simulations and in the TReno experiment is slightly different than the FACK version presented at Sigcomm96 [MM96a]. We replaced "Overdamping" and "Rampdown" by a combined "Rate-Halving" algorithm, which preserves the best properties of each. Rate-Halving quickly finds the correct window size following packet loss, even under adverse conditions, while maintaining TCP's Self-clock. In addition, we strengthen the retransmission strategy by decoupling it completely from congestion control considerations during recovery. An algorithm we call "Thresholded Retransmission" moves the *tcprexmtthresh* logic to the SACK scoreboard and applies it to every loss, not just the first. We also add "Lost Retransmission Detection" to determine when retransmitted segments have been lost in the network.

Rate-Halving congestion control adjusts the window by sending one segment per two ACKs for exactly one round trip during recovery. This sets the new window to exactly one-half of the data which was actually held in the network during the lossy round trip. At the beginning of the lossy round trip *snd.cwnd* segments have been injected into the network. Given that there have been some losses, we expect to receive (*snd.cwnd−loss*) acknowledgments. Under Rate-Halving we send half as

want to transfer a few Gigabytes of data at one time.

many segments, so the net effect on the congestion window is:

$$snd.cwnd = \left\lfloor \frac{snd.cwnd - loss}{2} \right\rfloor \qquad (15)$$

This algorithm can remain in Congestion Avoidance, without timing out, at higher loss rates than algorithms that wait for half of the packets to drain from the network when the window is halved.

We detect when exactly one round trip has elapsed by comparing the value of the forward-most SACK block in each ACK to the value of *snd.nxt* saved at the time the first SACK block arrived.

Bounding-Parameters add additional controls to guarantee that the final window is appropriate, in spite of potential pathological network or receiver behaviors. For example, a TCP receiver which sends superfluous ACKs could cause Rate-Halving to settle upon an inappropriately large window. Bounding-Parameters assure that this and other pathologies still result in reasonable windows. Since the Bounding-Parameters have no effect under normal operation, they have no effect on the results in this paper.

We are continuing to tinker with some of the details of these algorithms, but mostly in areas that have only minute effects on normal bulk TCP operations. The current state of our TCP work is documented at `http://www.psc.edu/networking/tcp.html`.

B Reaching Equilibrium

In several of our simulations and measurements we noted that an excessive amount of time was sometimes required for TCP to reach equilibrium (steady-state).

One interpretation of Equation 3 is that the average window size in packets will tend to $C/\sqrt{p}$. However, during a Slow-start (without Delayed ACKs), the expected window size is on the order of $1/p$ when the first packet is dropped, $2/p$ when the loss is detected, and back down to $1/p$ by the end of recovery (assuming SACK TCP). This window is much too large if p is small. It then takes roughly $\log_2\left(\frac{1/p}{C/\sqrt{p}}\right)$ congestion signals to bring the window down to the proper size. This requires the delivery of $\frac{1}{p}\log_2\frac{1}{C\sqrt{p}}$ packets, which is large if p is small.

The effect of this overshoot can be significant. Suppose $p = 1/256$ (approximately 0.004) then we have $1/\sqrt{p} = 16$ and $\log_2 1/\sqrt{p} = 4$. So it takes roughly 1000 packets to come into equilibrium. At 1500 bytes/packet, this is more than 1.5 Mbytes of data.

The average window in steady state will be 16 packets (24 kbytes). If the path has a 100 ms RTT, the steady state average bandwidth will be close to 2 Mb/s. However the peak window and bandwidth might be larger by a factor 16: 256 packet (6 Mbytes) and 30 Mb/s. (This is a factor of $\frac{1/p}{1/\sqrt{p}}$). The overshoot will be this pronounced only if it consumes a negligible fraction of the bottleneck link. Clearly this will not be the case over most Internet paths so the Slow-start will drive up the loss rate (or run out of receiver window) causing TCP to converge more quickly. It is unclear how significant this overshoot is in the operational Internet.

In all of our simulations we estimate the time required for the connection to reach steady-state, and exclude the initial overshoot when measuring loss, delay and bandwidth.

References

[B+97] Robert Braden et al. Recommendations on Queue Management and Congestion Avoidance in the Internet, March 1997. Internet draft draft-irtf-e2e-queue-mgt-00.txt (Work in progress).

[BOP94] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *Proceedings of ACM SIGCOMM '94*, August 1994.

[CH95] David D. Clark and Janey C. Hoe. Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes. Technical report, Internet End-to-End Research Group, 1995. Presentation. Cited for acknowledgement purposes only.

[CJ89] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[Cla96] Dave Clark. Private communication, December 1996. Derivation of Bandwidth vs. Loss.

[DLY95] Peter B. Danzig, Zhen Liu, and Limim Yan. An Evaluation of TCP Vegas by Live Emulation. *ACM SIGMetrics '95*, 1995.

[FF96] Kevin Fall and Sally Floyd. Simulations-Based Comparisons of Tahoe, Reno and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[FJ92] Sally Floyd and Van Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[Flo91] Sally Floyd. Connections with Multiple Congested Gateways in Packet-Switched

Networks, Part 1: One-way Traffic. *Computer Communications Review*, 21(5), October 1991.

[Flo95] Sally Floyd. TCP and Successive Fast Retransmits, February 1995. Obtain via ftp://ftp.ee.lbl.gov/papers/fastretrans.ps.

[Flo96] Sally Floyd. SACK TCP: The sender's congestion control algorithms for the implementation sack1 in LBNL's ns simulator (viewgraphs). Technical report, TCP Large Windows Working Group of the IETF, March 1996. Obtain via ftp://ftp.ee.lbl.gov/talks/sacks.ps.

[Hoe95] Janey C. Hoe. Startup Dynamics of TCP's Congestion Control and Avoidance Schemes. Master's thesis, Massachusetts Institute of Technology, June 1995.

[Hoe96] Janey C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. *Proceedings of ACM SIGCOMM '96*, August 1996.

[Jac88a] Van Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM '88*, August 1988.

[Jac88b] Van Jacobson. Traceroute Source Code, 1988. Obtain via ftp from ftp.ee.lbl.gov.

[Jac90] Van Jacobson. Modified TCP Congestion Avoidance Algorithm. Email to end2end-interest Mailing List, April 1990. Obtain via ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[JB88] Van Jacobson and Robert Braden. TCP Extensions for Long-Delay Paths, October 1988. Request for Comments 1072.

[JBB92] Van Jacobson, Robert Braden, and Dave Borman. TCP Extensions for High Performance, May 1992. Request for Comments 1323.

[LM94] T.V. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IFIP Transactions C-26, High Performance Networking*, pages 135–150, 1994.

[Mat94a] Matthew Mathis. Private communication, November 1994. Derivation of Bandwidth vs. Loss.

[Mat94b] Matthew B. Mathis. Windowed Ping: An IP Layer Performance Diagnostic. *Proceedings of INET'94/JENC5*, 2, June 1994.

[Mat96] Matthew Mathis. Diagnosing Internet Congestion with a Transport Layer Performance Tool. *Proceedings of INET'96*, June 1996.

[Mat97] Matthew Mathis. Internet Performance and IP Provider Metrics information page. Obtain via http://www.psc.edu/~mathis/ippm/, 1997.

[MF95] Steven McCanne and Sally Floyd. ns–LBL Network Simulator. Obtain via: http://www–nrg.ee.lbl.gov/ns/, 1995.

[MM96a] Matthew Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. *Proceedings of ACM SIGCOMM '96*, August 1996.

[MM96b] Matthew Mathis and Jamshid Mahdavi. TCP Rate-Halving with Bounding Parameters, October 1996. Obtain via: http://www.psc.edu/networking/papers/FACKnotes/current/.

[MMFR96] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. Request for Comments 2018.

[OKM96a] Teunis Ott, J.H.B. Kemperman, and Matt Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. In progress, August 1996. Obtain via pub/tjo/TCPwindow.ps using anonymous ftp to ftp.bellcore.com, See also [OKM96b]., August 1996.

[OKM96b] Teunis J. Ott, J.H.B. Kemperman, and Matt Mathis. Window Size Behavior in TCP/IP with Constant Loss Probability, November 1996.

[Ost96] Shawn Ostermann. tcptrace TCP dumpfile analysis tool. Obtain via http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html, 1996.

[Pax97a] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. *Proceedings of ACM SIGCOMM '97*, August 1997.

[Pax97b] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

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

[Ste97] W. Richard Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. Request for Comments 2001.