The performance of TCP/IP for networks with high bandwidth-delay products and random loss

T. V. Lakshman Upamanyu Madhow *

Abstract

This paper examines the performance of TCP/IP, the Internet data transport protocol, over Wide Area Networks (WANs) in which data traffic could coexist with real-time traffic such as voice and video. Specifically, we attempt to develop a basic understanding, using analysis and simulation, of the properties of TCP/IP in a regime where (1) the bandwidth-delay product of the network is high compared to the buffering in the network, and (2) there may be transient congestion due to fluctuations in real-time traffic, modeled here as producing random losses among the packets of the TCP connection of interest. The following key results are obtained. First, random loss leads to significant throughput deterioration when the product of the loss probability and the *square* of the bandwidth-delay product is larger than one. Unless network resources are specifically reserved for data traffic, data traffic will inevitably incur random losses due to transient fluctuations in higher priority real-time traffic when the network is highly utilized. Second, for multiple connections sharing a bottleneck link, TCP is grossly unfair towards connections with higher round-trip delays. This means that a simple First In First Out (FIFO) queueing discipline might not suffice for data traffic in WANs. Finally, we observe that, while the recent Reno version of TCP produces less bursty traffic than the original Tahoe version, it is less robust than the latter when successive losses are closely spaced. We conclude by indicating modifications that may be required both at the transport and network layers to provide good end-to-end performance over high-speed WANs.

1 Introduction

Most existing data transfer protocols have been designed for Local Area Network (LAN) applications in which buffer sizes far exceed the bandwidth-delay product. However, the interconnection of LANs using high-speed backbone networks has resulted in Wide Area Networks (WANs) in which the buffering available to a connection at switches and routers may be smaller than the bandwidth-delay product of the connection[1]. In addition, in the Internet of the future, data traffic will share the network with voice and video traffic. In this paper, we examine the performance of the most popular data transfer protocol in current use, TCP/IP, in the preceding scenario. This is essential not only for network provisioning in the short term (since the rapid growth of Web applications has caused TCP traffic to grow correspondingly) but also for determining how TCP needs to be modified in the longer term.

We study two versions of TCP: one is the popular Tahoe version developed by Jacobson in 1988 [11] (henceforth called *TCP-tahoe*); the other is the Reno version, which includes the fast retransmit option together with a method for reducing the incidence of slow start, suggested by Jacobson in 1990 [12] (we will refer to this as *TCP-reno*). We attempt to develop a basic understanding of these schemes by considering one-way traffic over a single bottleneck link with FIFO transmission. For Local Area Networks, the round-trip delay of the connection is small, so that the bandwidth-delay product could be much smaller than the buffering on the bottleneck link. We are more interested, however, in WANs with large round-trip delays, so that the buffering on the bottleneck link is typically of the same order of magnitude as, or smaller than, the bandwidth-delay product (this is what we mean by *high bandwidth-delay products* throughout this paper). While our approximate analysis applies to either situation, our numerical results comparing analytical predictions with simulations are mostly restricted to the high bandwidth-delay product regime. The bottleneck link may be shared by several TCP connections. In addition, we also assume that each packets may be lost randomly even after obtaining service at the bottleneck link.

Random loss is a simple model for a scenario of particular interest in the context of networks with multimedia traffic, where transient fluctuations in real time traffic may cause irregularly spaced losses for data traffic. This would occur, for instance, for both the UBR and ABR service classes [1] in ATM networks. The only difference is that for ATM ABR, each connection would have a time-varying available rate determined by feedback from the switches, so that most random losses would occur at the interface of the source to the network, since that is where the available rate would be enforced. In addition to serving as a model for transient congestion, we note that random loss on the Internet has been reported [3], where it is conjectured to occur due to a variety of reasons, including intermittent faults in hardware elements such as Ethernet/FDDI adapters, and incorrect handling of arriving packets by routers. Finally, with the anticipated emergence of mobile computing over heterogeneous net-

*T. V. Lakshman is with the High Speed Networks Research Dept., Bell Laboratories, Lucent Technologies, 101 Crawfords Corner Road, Holmdel , NJ 07733. Upamanyu Madhow is with the ECE Dept. and the Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801, and his work was supported in part by the U. S. Army Research Office under grant DAAH04-95-1-0246.

[1] The *bandwidth-delay product* is loosely defined to be the product of the round-trip delay for the connection and the capacity of the bottleneck link in its path.

works with both wireless and wireline links, losses and time variations due to wireless links in the path of the connection can also be accomodated via a random loss model. Since our purpose is to obtain a fundamental understanding of TCP, none of the preceding situations are explicitly considered in this paper. However, as discussed in Section 6, the results here should provide a basis for further work on developing network level design guidelines for supporting TCP. In particular, in recent work [5], we have built upon these results to develop analytical guidelines for the design of wireless-wireline interfaces for supporting TCP over heterogeneous networks.

One of the simplifications of the model used for our analysis is that two-way traffic is not considered. Feedback systems are notoriously difficult to analyze, so that even our simple model is not amenable to exact analysis. However, not only does our approximate analysis match simulation results for the idealized system model, but it also provides a close match to results for a detailed simulation that includes two-way traffic for multiple TCP-reno connections over an ATM network (described in Section 5). The latter is used to eliminate the possibility that phase effects are an artifact of a simple deterministic model. Thus, despite its simplicity, our model serves to bring out the fundamental characteristics of the dynamic window mechanism used in TCP. Two-way traffic would only produce further performance deterioration due to the phenomenon of *ack compression* [27] resulting from the queueing of acknowledgements.

We obtain the following key results. Discussion of the implications of these results for system design is postponed to Section 6.

(1) While TCP-reno produces less bursty traffic than TCP-tahoe, it is much less robust towards "phase effects". The latter term refers to unpredictability in performance resulting from very small differences in the relative timings of packet arrivals for different connections sharing a link.

(2) Both versions of TCP appear to have significant drawbacks as a means of providing data services over multimedia networks, because random loss resulting from fluctuations in real-time traffic can lead to significant throughput deterioration in the high bandwidth-delay product regime. Roughly speaking, the performance is degraded when the product of the loss probability and the *square* of the bandwidth delay product is large (e.g., ten or more).

(3) For high bandwidth-delay products, TCP is grossly unfair towards connections with higher propagation delays: for multiple connections sharing a bottleneck link, the throughput of a connection is inversely proportional to (a power of) its propagation delay.

It is worth clarifying that random loss causes performance deterioration in TCP because it does not allow the TCP window to reach high enough levels to permit good link utilization. On the other hand, when the TCP window is already large and is causing congestion, random early drops of packets when the link buffer gets too full can actually enhance performance and alleviate phase effects [10].

Early simulation studies of TCP-tahoe include [24], [26], [27]. Simulations for the simple multi-hop network considered in [26] showed the oscillations in window sizes and the unfairness of TCP towards connections traversing a larger number of hops. In [24], the authors consider a number of TCP connections sharing a bottleneck link. There is no queueing of acknowledgements, and sources are assumed to always have data to send. As mentioned previously, [27] considers the effect of two-way traffic. While our model is similar to that used in [24], the key differences between our paper and these three previous studies are that (a) the ratio of bandwidth-delay product to buffer size is much higher in our study, (b) the effect of random loss due to transient congestion (or other sources) is included. Thus, some of the undesirable features of TCP-tahoe which arise specifically for networks with high bandwidth-delay products (such as excessive buffering requirements and vulnerability to random loss) were not noticed in these earlier studies. Furthermore, in contrast to these previous studies, we place much more emphasis on detailed analytical insight on the effects of various parameters on performance. It is difficult to derive anything more than qualitative insights from simulation alone, since the quantitative results are only for a particular set of parameter values.

The bias of TCP-tahoe against connections with large round-trip delays and against connections traversing a large number of congested gateways has also been brought out in other recent studies of TCP-tahoe [8], [9]. A heuristic analysis in [8] shows that, for multiple connections sharing a bottleneck link, the throughput of a connection is inversely proportional to its round-trip time. While we consider a similar system in Section 5, our analysis is more detailed, taking explicit account of the buffer size and the bandwidth-delay product. Oscillatory behavior and unfairness towards connections with larger propagation delays have also been noticed in a previous analytical study of feedback-based congestion control [2] which uses a continuous-time approximation to the dynamic behavior of a rate-based scheme. Other analyses of flow control schemes include [22], [23], and [20], but these references do not address the specific concerns raised here in any detail.

An alternative adaptive window flow control scheme that is of particular interest in the light of our findings is that proposed in [18], [19]. The window adaptation mechanism there is designed to operate in a high bandwidth-delay product regime, and is based on asymptotics derived from a queueing model of the network. This scheme has the drawback of requiring more centralized coordination than TCP: the adaptation algorithm for each connection must know the (time-varying) identity of the bottleneck link for that connection and the relative propagation delays of the other connections sharing that link. However, the adaptive mechanism itself is much smoother (in the sense that the window size reaches and oscillates around an equilibrium value, rather than probing for bandwidth until there is a loss) than the drastic window size changes in TCP, so that a decentralized adaptive scheme based on a similarly smooth mechanism may overcome the drawbacks in TCP while not requiring the kind of knowledge assumed in [18], [19]. Another possible mechanism is adaptation based on explicit feedback. Examples are the DECbit protocol [22] (which suffers from some of the same problems as TCP) and the mechanisms recently proposed by the ATM Forum [1] for its ABR service class. Much further thought, however, is required as to the best way to obtain and utilize explicit feedback in networks with high bandwidth-delay product.

The system model is described in Section 2. Analytical and simulation results for the evolution of a single connection in the absence of random loss are given in Section 3. Section 4 considers the effect of random loss. Section 5 contains results for multiple connections with and without random loss. We give our conclusions in Section 6.

2 System Model

Our model is similar to that considered in [24] and [27]. We consider infinite data sources which always have packets to send, so that the units of data are maximum sized packets (in general, packet sizes in TCP may be variable). We consider a single bottleneck link with capacity μ packets per second and a FIFO buffer of size B packets, as shown in Figure 1. Any packet arriving when the buffer is full is lost (random loss may cause additional losses). The number of connections, or sources, sharing the link is assumed to be constant. For each connection, all delays except for service time and queueing at the bottleneck link are lumped into a single "propagation delay," which includes (a) the time between the release of a packet from the source and its arrival into the link buffer, (b) the time between the transmission of the packet on the bottleneck link and its arrival at its destination, and (c) the time between the arrival of the packet at the destination and the arrival of the corresponding acknowledgement at the source. The propagation delay for a packet from the ith connection is denoted by τ_i.

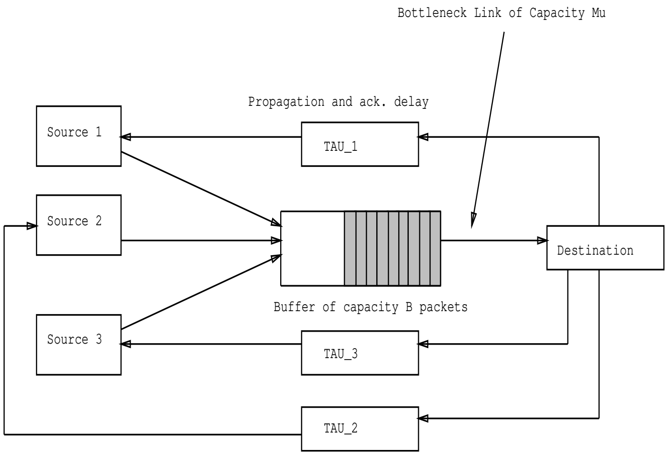

Figure 1: System model showing three connections

The τ_i are taken to be deterministic, which implicitly assumes that deterministic propagation and processing delays are more significant than random queueing delays at all nodes and links other than the bottleneck link. Although such an assumption is overly simplistic even for a relatively simple system with two-way traffic [27], it suffices for our present purpose of arriving at a basic understanding of the interaction between different connections sharing a link.

Each connection is assumed to use a window flow control protocol. At time t, the window size for connection i is denoted by $W_i(t)$, and is equal to the maximum allowed number of unacknowledged packets (retransmissions are not counted). It is assumed that each connection uses its allowable window to the fullest extent, i.e., that at time t, there are indeed $W_i(t)$ unacknowledged packets for connection i. The window varies dynamically in response to acknowledgements and detection of packet loss. Upon receiving a packet, the destination is assumed to send an acknowledgment back immediately. These acknowledgments are *cumulative* and indicate the next byte expected by the receiver. We have implicitly assumed that the processing delay at the receiver is accounted for in the propagation delay, and that variations in this delay are negligible relative to the magnitude of the τ_i. This may not be true in practice (at least for smaller bandwidth-delay products), and the effect of such variations on performance requires further study.

In the original version of TCP-tahoe, packet loss is detected by maintaining a timer based on an estimate of the round-trip time. When a packet is sent, a timeout value is computed using the current round-trip time estimate and the timer is started. Expiry of this timer is taken to signal packet loss. For each retransmission following a timer expiry, the timer value used is twice the previous timer value. Estimates of the round-trip time are obtained by measuring the round-trip time upon receipt of unambiguous acknowledgements (i.e. ignoring acknowledgements for retransmitted segments) and computing a weighted average of the old and new estimates. Refer [15, 25] for a detailed description of round-trip time estimation. We will refer to a timer based on this estimate as a *fine-grained timer*, in order to distinguish it from the *coarse-grained timers* used in practice, which are typically multiples of 500 milliseconds. In order to prevent a needlessly lengthy stoppage of transmission upon expiry of a coarse-grained timer, most current versions of both TCP-tahoe and TCP-reno incorporate a *fast retransmit* option: if the number of *duplicate acknowledgements* (i.e., multiple acknowledgements with the same "next expected" packet number n) exceeds a threshold, packet n is assumed to be lost. In this paper, we implement fine-grained timers in our simulations, in order to study the dynamic evolution of TCP (and to highlight possible shortcomings) in the most ideal setting. The original version of TCP-tahoe, without fast retransmit, is implemented. However, in simulation results not reported here, we have checked that coarse-grained timers with fast retransmit give virtually identical performance in most cases of interest for TCP-tahoe (unless almost all packets in a window are lost, fast retransmit detects loss very effectively). For TCP-reno, we implement fast retransmit with a fine-grained timer in our simulations. Because TCP-reno has a less robust congestion control mechanism, we have found in later work that the use of a coarse-grained timer does impact its performance even with fast retransmit, unlike for TCP-tahoe. Since either fine-grained timers or the fast retransmit option provide almost perfect loss detection, it is assumed in our analysis that packet losses are detected perfectly.

A simplified description of TCP-tahoe [11] and TCP-reno [12] follows.

Description of TCP-tahoe

The algorithm followed by each connection has two parameters, current window size W and a threshold W_t, which are updated as follows.

(1) After every acknowledgement of a new packet:
if $W < W_t$, set $W = W + 1$; **Slow Start Phase**
else set $W = W + 1/[W]$. **Congestion Avoidance Phase**
($[x]$ denotes the integer part of x)
(2) After a packet loss is detected:
set $W_t = W/2$;
set $W = 1$.

The algorithm typically evolves as follows (although, as described in the next section, the evolution is somewhat different for relatively small buffer size): when packet loss is detected, the window is reduced to one. In the slow start phase that follows, the window grows rapidly for every successfully acknowledged packet until it reaches half of the window size at the last packet loss. The algorithm then switches to the congestion avoidance phase, probing for extra bandwidth by incrementing the window size by one for every window's worth of acknowledged packets. This growth continues until another packet loss is detected, at which point another cycle begins. We use the term *cycle* to mean TCP evolution starting from the end of one congestion avoidance phase to the end of the next. In Section 3, it turns out that, for our simple model, TCP evolution is periodic if there is no random loss, so that successive cycles are identical. In Section 4, on the other hand, where we consider random loss, the duration of, and window evolution within, different cycles is random.

Description of TCP-reno

After the number of duplicate acknowledgements exceeds a threshold (typically three), TCP-reno retransmits the packet, but instead of cutting the window back to one, it only reduces it by a factor of two. However, in order to prevent a burst of packets from being transmitted when the retransmission is finally acknowledged, it temporarily permits new packets to be transmitted with each repeated acknowledgement until the "next expected" number in the acknowledgement advances. While these subtleties are essential to the working of the algorithm (see [12] for details) and are implemented in our simulations, the following simplified description is adequate for conveying an understanding of the algorithm's behavior.

(1) After every non-repeated acknowledgement, the algorithm works as before:
if $W < W_t$, set $W = W + 1$; **Slow Start Phase**
else set $W = 1 + 1/[W]$. **Congestion Avoidance Phase**

(2) When the duplicate acknowledgements exceeds a threshold,
retransmit "next expected" packet;
set $W_t = W/2$, then set $W = W_t$ (i.e. halve the window);
resume congestion avoidance using new window once retransmission is acknowledged.

(3) Upon timer expiry, the algorithm goes into slow start as before:
set $W_t = W/2$;
set $W = 1$.

In this case, after an initial slow start transient, the typical cyclical evolution does not involve slow start, since the window size is halved upon loss detection. Each cycle begins when a loss is detected via duplicate acknowledgements. Assuming that loss occurs at window size W_{max}, the window size at the beginning of each cycle is $W_{max}/2$. The algorithm resumes probing for excess bandwidth in congestion avoidance mode until the window size reaches W_{max} again, at which point a loss occurs and a new cycle with window size $W_{max}/2$ begins. We will show that the throughput attained by this scheme is higher than that of TCP-tahoe, especially when the buffer size is small compared to the bandwidth-delay product. However, this algorithm is almost as vulnerable to random loss.

For the remainder of this paper, we will use W_{max} as a generic notation for the window size at which congestion avoidance ends. The value of W_{max} could therefore change from cycle to cycle if loss occurs randomly, or could be the same for all cycles if loss occurs periodically. It is worth relating our notation to that usually used in TCP code (see [24], for instance): W is usually referred to as the congestion window *cwnd*, and W_t is denoted by *ssthresh*. The actual window for flow control purposes is taken to be the minimum of *cwnd* and *maxwnd*, where the latter is set by the receiver. For the purpose of this paper, the window size is assumed to be dictated by the capacity and buffering of the bottleneck link (i.e., $cwnd \leq maxwnd$), so the actual window size equals the congestion window. Note that some form of window scaling (i.e., increasing the window size in bytes while using the same sequence number space, by scaling up the size of the data segment referred to by a given number) may be required to achieve this for large bandwidth-delay products [14].

3 Evolution without Random Loss

We consider the evolution of a single connection and derive expressions for its long-term throughput. It is convenient to define the normalized buffer size $\beta = \frac{B}{\mu\tau+1} = \frac{B}{\mu T}$, where τ denotes the propagation delay for each packet of the connection and $T = \tau + 1/\mu$ denotes the propagation delay plus the service time. Since we are concerned with large bandwidth-delay products, we restrict attention to $\beta \leq 1$ in this section. In contrast, simulations in earlier work [24] consider $\beta \gg 1$, for which the average throughput is close to the capacity of the bottleneck link. For brevity, expressions for the latter case are omitted.

The maximum window size that can be accommodated in steady state in the bit pipe is

$$W_{pipe} = \mu T + B = \mu\tau + B + 1 \qquad (1)$$

In this case, the buffer is always fully occupied and there are μT packets in flight. The cyclical evolution of TCP-tahoe consists of a slow start phase starting with $W = 1$ and continuing until the window size reaches $W_t = W_{pipe}/2$, followed by congestion avoidance until

Time	Packet Acked	Window Size	Packet(s) Released	Queue Length
0		1	1	1
T	1	2	2, 3	2
$2T$	2	3	4, 5	2
$2T+1/\mu$	3	4	6, 7	$2-1+2=3$
$3T$	4	5	8, 9	2
$3T+1/\mu$	5	6	10, 11	$2-1+2=3$
$3T+2/\mu$	6	7	12, 13	$2-1+2-1+2=4$
$3T+3/\mu$	7	8	14, 15	$2-1+2-1+2-1+2=5$
$4T$	8	9	16, 17	2
.				
.				
.				

Table 1: Evolution during slow start phase

$W = W_{pipe}$. The next increase in window size leads to buffer overflow, at which point the window is reset to one and a new cycle starts. We will show that if the relative buffer size β is not large enough, buffer overflow may occur even in the slow start phase, and the cyclical evolution is somewhat different from the preceding description. For TCP-reno, if the scheme functions as designed, slow start is eliminated from the cyclical evolution. In each cycle, the algorithm starts from $W = W_t = W_{pipe}/2$, does congestion avoidance until $W = W_{pipe}$, and drops back to $W = W_t = W_{pipe}/2$ after a packet loss due to buffer overflow is detected via duplicate acknowledgements.

In each case, if the number of packets successfully transmitted during a cycle is N_c and the duration of a cycle is T_c, then the periodic evolution implies that the average throughput is given by $\bar{\lambda} = N_c/T_c$. In the following, we describe this evolution more carefully, and compute these quantities in sufficient detail to produce an excellent match with simulations (see Table 2)

3.1 Slow Start Phase

The slow start phase must be considered in some detail to understand the advantage of TCP-reno over TCP-tahoe. Starting from $W = 1$ with slow start threshold W_t, the window size is increased by one for every acknowledgement in this phase, so that two packets are released into the buffer for every acknowledgement. Table 1 shows the evolution of the window size and the queue length in this phase. For every acknowledgement, we indicate the number of the packet which was acknowledged (for convenience, we number the packets in increments of one rather than in increments equal to the number of bytes per packet).

The evolution in Table 1 is best described by considering *mini-cycles* of duration equal to the round-trip time T, where the ith mini-cycle refers to the time interval $[iT, (i+1)T)$ (the mini-cycles are separated by lines in the table). The acknowledgement for a packet released in mini-cycle i arrives in mini-cycle $(i+1)$, and increases the window size by one. This leads to a doubling of the window in each mini-cycle. Further, acknowledgements for consecutive packets served in mini-cycle i arrive spaced by the service time $1/\mu$ during mini-cycle $(i+1)$, and two packets are released for each arriving acknowledgement, leading to a buildup of queue size. The preceding evolution assumes implicitly that the normalized buffer size $\beta < 1$, so that the window size during the slow start phase is smaller than μT and the queue empties out by the end of each mini-cycle. Denoting the window size at time t by $W(t)$, we obtain that, during the $(n+1)$th mini-cycle,

$$W(nT + m/\mu) = 2^n + m + 1, \quad 0 \le m \le 2^n - 1 \qquad (2)$$

where we have assumed that $(2^n - 1)/\mu < T$. Similarly, letting $Q(t)$ denote the queue length at time t, the queue build-up during the nth mini-cycle is given by

$$Q(nT + m/\mu) = m + 2, \quad 0 \le m \le 2^n - 1 \qquad (3)$$

The maximum queue length during the $(n+1)$th mini-cycle is therefore $2^n + 1$, which is approximately half the maximum window size $W(nT + (2^n - 1)/\mu) = 2^{n+1}$ during that mini-cycle. For a buffer size B, we can use (2) and (3) to determine the window size at which , the queue length exceeds the buffer size as follows. Define the integer $n_b = \lceil log_2(B-1) \rceil$, so that $2^{n_b - 1} + 1 < B \le 2^{n_b} + 1$. From (3), buffer overflow will occur in the n_bth mini-cycle (the largest queue length in the previous cycle is $2^{n_b - 1} + 1$, which is smaller than B), with $m + 2 = B + 1$. From (2), the window size W_b at which this happens is $2^{n_b} + m + 1$, so that that

$$W_b = 2^{n_b} + B \qquad (4)$$

Buffer overflow during a slow start phase with threshold W_t thus occurs only if

$$W_b \le W_t \qquad (5)$$

A more explicit condition for buffer overflow can be derived as follows. Assuming that the packet loss causing the slow start phase occurred when the window size exceeds the value $W_{pipe} = \mu T + B$, the slow start threshold equals $W_t = W_{pipe}/2 = (1+\beta)\mu T/2$. Since $W_b \approx 2B = 2\beta\mu T$, the condition for buffer overflow (5) is approximately equivalent to $\beta \le 1/3$. As shown by our tables of throughput as a function of β later in this section, for TCP-tahoe, there is a clear discontinuity in throughput around $\beta \approx 1/3$, since having two slow start phases instead of one reduces the throughput.

Figure 2 shows the simulated congestion window and buffer occupancy evolution for a single connection using TCP-tahoe with $\tau = 1$, $\mu = 100$ and $\beta = .1$ (i.e., $B = 10$). The congestion window size is shown by the solid line and the buffer occupancy by the dotted line. As expected, the window grows to $W_{pipe} = \mu\tau + B + 1 = 111$ and the next increase in the window causes a packet to be dropped. Detection of this loss (upon expiry of the associated timer) causes the window to be reduced to one and initiation of the slow start phase. The figure clearly shows the rapid growth in window size during the slow start phase. However, since $\beta < 1/3$, buffer overflow occurs when $W = W_b = 26$, and is detected by the time the window size reaches $W = 2W_b - 2 = 50$ (see the discussion later in this section). A second slow start phase

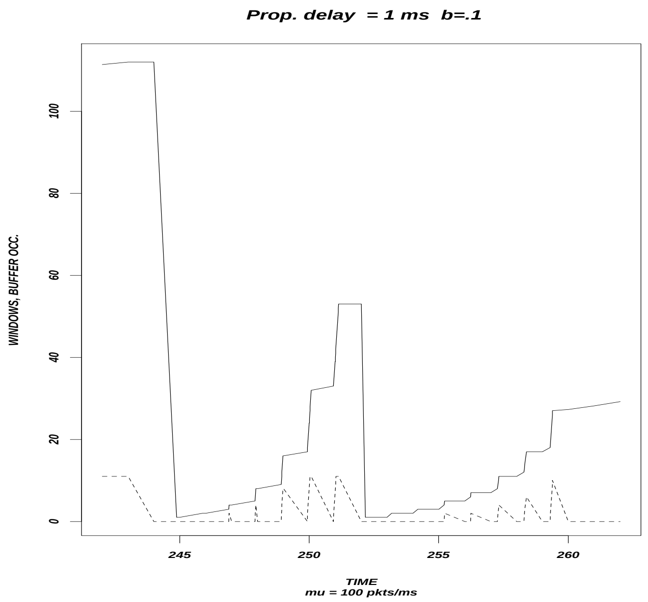

Figure 2: Window and buffer evolution for a single connection: Two slow starts

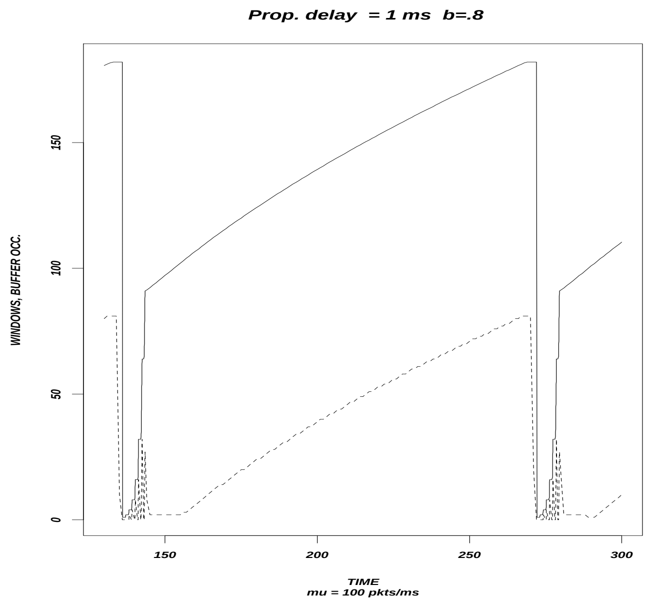

Figure 3: Window and buffer evolution for a single connection: one slow start

is initiated at this point with threshold 25. This window size is reached without further loss, at which point slower window growth due to congestion avoidance commences. This lasts until the window exceeds $W_{pipe} = 111$, after which a new cycle begins.

When β is greater than 1/3 the window evolution for TCP-tahoe is quite different. This is illustrated in Figure 3 which shows the evolution of window sizes and buffer occupancy for $\beta = .8$. Here, packet loss is seen to occur when the window is of size $W_{pipe} = 181$. As before, detection of this loss causes the window size to be reduced to one and initiates slow start. However, there is no packet loss in the slow start phase, which terminates when the window reaches 91. In the congestion avoidance phase that follows, the window grows linearly and then more slowly (as explained in the next subsection) until the window exceeds W_{pipe}. This results in a packet loss causing the cycle to repeat. The absence of the double slow start results in much higher throughput, since the initial window size for the congestion avoidance phase (which accounts for most of the packets transmitted) is higher.

We now compute the duration and number of packets transmitted during the slow start phase(s) in a given cycle for TCP-tahoe. Even though many subtleties in timing are glossed over, the computations are accurate enough to result in excellent agreement with the throughput obtained by simulations. It is worth repeating that slow start does not occur in the cyclical evolution of TCP-reno, although it may be used when the connection is originally set up.

Case 1: $W_b > W_t$

There is only one slow start phase per cycle, which ends when the window size reaches $W_t = W_{pipe}/2$. We use a simplified version of (2), $W(t) = 2^{t/T}$ to approximate the duration of this phase as

$$t_{ss} = Tlog_2 W_t \tag{6}$$

The number of packets transmitted in this phase is approximated by

$$n_{ss} = W_t \tag{7}$$

(Since the window size grows by one for every acknowledgement during slow start, starting from an initial value of one, the number of packets transmitted successfully during slow start is well approximated by the window size at the end of the slow start period.)

Case 2: $W_b \leq W_t$

There are two slow start phases in a cycle in this case. Let t_{ss1} denote the duration of the first slow start phase and let n_{ss1} denote the number of packets successfully transmitted during that phase. Let t_{ss2}, n_{ss2} denote the analogous quantities for the second slow start phase. Computation of average throughput requires the computation of these quantities.

In the first slow start phase (with threshold $W_t = W_{pipe}/2$), a buffer overflow occurs when the window size reaches W_b. The duration of this phase is the time taken to reach W_b, approximated by $Tlog_2 W_b$, together with the time taken to detect the loss, which is taken to be one round-trip time T, so that

$$t_{ss1} = T(log_2 W_b + 1) \tag{8}$$

The number of packets transmitted in this phase is, reasoning as before, taken to be

$$n_{ss1} = W_b \tag{9}$$

Since the buffer overflow in the first slow start phase is not detected till the mini-cycle after which it happens, the window size at which the overflow is detected can be shown, by means of a more careful analysis, to be approximately $W^* \approx \min\{2W_b - 2, W_t\}$. This implies that the threshold for the second slow start phase is

$$\tilde{W_t} = W^*/2 \approx \min\{W_b - 1, W_t/2\}. \tag{10}$$

The duration and number of packets transmitted during the second slow start phase is now obtained as in Case 1 to be

$$t_{ss2} = T log_2 \tilde{W}_t, \qquad (11)$$

$$n_{ss2} = \tilde{W}_t. \qquad (12)$$

The total time spent in slow start and the number of packets transmitted during slow start are then given by

$$t_{ss} = t_{ss1} + t_{ss2}, \quad n_{ss} = n_{ss1} + n_{ss2}. \qquad (13)$$

3.2 Congestion Avoidance Phase

We can unify the analysis of this phase for TCP-tahoe and TCP-reno by assuming that the congestion avoidance phase starts from an arbitrary window size W_0 and terminates when the window size reaches W_{max}. In all cases considered in this section, $W_{max} = W_{pipe}$. For TCP-tahoe, W_0 equals the slow start threshold for the slow start phase immediately preceding the congestion avoidance phase, and is given by

$$W_0(tahoe) = \begin{cases} W_{max}/2, & W_{max}/2 \ < \ W_b, \\ \min\{W_b - 1, W_{max}/4\}, & W_{max}/2 \ \geq \ W_b. \end{cases} \qquad (14)$$

Remark: For $W_{max}/2 \geq W_b$, there are two slow start phases, so that the window size at the beginning of congestion avoidance is the slow start threshold for the second slow start phase, which is given in (10).

For TCP-reno, we have

$$W_0(reno) = W_{max}/2, \qquad (15)$$

since the window size is halved after losing a packet.

In contrast to the slow start phase, the congestion avoidance phase, by virtue of its slower window growth, is well-modeled by a continuous-time approximation for window evolution. Such approximations have been used previously in [24] to explain simulation results. Let $\frac{dW}{dt}$ denote the rate of window growth with time, $\frac{dW}{da}$ the rate of window growth with arriving acknowledgements, and $\frac{da}{dt}$ the rate at which the acknowledgements are arriving. Then, during the congestion avoidance phase,

$$\frac{dW}{da} = 1/W. \qquad (16)$$

The acknowledgements arrive back at a rate equal to the instantaneous throughput λ, so that

$$\frac{da}{dt} = \lambda = \min\ \{W/T, \mu\} \qquad (17)$$

Combining (16) and (17), we obtain that

$$\frac{dW}{dt} = \begin{cases} 1/T, & W \leq \mu T, \\ \mu/W, & W \geq \mu T. \end{cases} \qquad (18)$$

Thus, for $W \leq \mu T$, the window W grows as t/T. The duration of this period of growth is therefore given by

$$t_A = T(\mu T - W_0), \qquad (19)$$

since the initial window size was W_0 (for $\beta < 1$, $W_0 \leq W_{max}/2$ is always less than μT). The number of packets transmitted during this time is given by

$$n_A = \int_0^{t_A} W(t + t_{ss})/T dt = \int_0^{t_A} (W_0 + t/T)/T dt$$
$$= [W_0 t_A + t_A^2/(2T)]/T \quad (20)$$

When $W > \mu T$, we obtain from (18) that W^2 grows as $2\mu t$, so that, for $t \geq t_A$, $W^2(t) = 2\mu(t - t_A) + (\mu T)^2$. This growth period, and the cycle, terminates with buffer overflow when the window size exceeds W_{max}, and its duration is given by

$$t_B = [W_{max}^2 - (\mu T)^2]/(2\mu) \qquad (21)$$

($t_B = 0$ if $W_{max} < \mu T$, although this does not occur for $W_{max} = W_{pipe} = \mu T + B$.) Since the bottleneck link is being fully utilized during this period, the number of packets transmitted is given by

$$n_B = \mu t_B, \qquad (22)$$

3.3 Throughput Computation and Numerical Results

Due to the periodic evolution, the long-run average throughputs for both TCP-tahoe and TCP-reno are equal to the average throughputs in a cycle, and are given by

$$\bar{\lambda} = (n_{ss} + n_A + n_B)/(t_{ss} + t_A + t_B), \quad \text{for tahoe} \quad (23)$$

$$\bar{\lambda} = (n_A + n_B)/(t_A + t_B), \quad \text{for TCP} - \text{reno}, \qquad (24)$$

where the preceding quantities are as computed in Sections 3.1 and 3.2.

Table 2 gives the link utilization as a function of the normalized buffer size β for both TCP-tahoe and TCP-reno. The results obtained using (23)-(24) are compared with those obtained using simulation for an example with high bandwidth-delay product. The match is within 2 %. For TCP-tahoe, a clear thresholding effect is seen at $\beta = .31$ (recall that the analysis predicted a thresholding effect around $\beta \approx \frac{1}{3}$). The utilization for TCP-reno is uniformly higher for all values of β, and there is no thresholding effect for small β. The difference in utilization is small for large β, since the congestion avoidance phase for the two schemes is identical, and the duration of the slow start phase is small compared to the duration of the cycle.

4 Evolution with Random Loss

We assume here that any given packet may be lost with probability q, and that these random losses are independent. As in the previous section, we consider a single connection, and show that, for both TCP-tahoe and TCP-reno, the throughput is strongly dependent on $q(\mu T)^2$, and deteriorates sharply compared to the lossless throughput when this quantity becomes large. Roughly speaking, the throughput degradation occurs because packet losses relatively early in a cycle result in small initial values for the congestion avoidance phase, in which the bulk of the packets are transmitted. This results in

β	Link Utilization (Analysis/Simulation)	
	Tahoe	Reno
.1	.604/.604	.818/.818
.2	.660/.664	.871/.870
.31	.708/.718	.915/.911
.32	.856/.858	.919/.916
.8	.953/.954	.996/.994

Table 2: Link Utilization as a function of normalized buffer size ($\mu = 100, \tau = 1$)

small window sizes (determined by random losses rather than congestion), and therefore low link utilizations, during the cycle.

In principle, it is possible to exactly compute the throughput based on a Markov chain analysis, and we sketch this method in the following. However, insight into when random loss causes throughput deterioration is better obtained by means of an approximate analysis, which we pursue in more detail.

In the absence of random loss, the evolution of a cycle in TCP-tahoe is completely determined by the slow start threshold W_t (which is half the window size at the end of the previous cycle). Similarly, the evolution of a cycle in TCP-reno is determined by the window size W_0 at the beginning of the cycle (again, this is half the window size at the end of the previous cycle). Since a single parameter w ($w = W_t$ for TCP-tahoe and $w = W_0$ for TCP-reno) determines the cyclical evolution, the following functions are well-defined for either scheme (although they may be hard to compute explicitly):

$W(n, w)$, the window size after n packets are successfully transmitted,

$T(n, w)$, the time taken for n packets to be successfully transmitted,

$N_{max}(w)$, the number of packets successfully transmitted before the cycle ends with buffer overflow.

If we now introduce random loss, the cycle may terminate due to a random loss after $N \leq N_{max}$ packets have been successfully transmitted. According to our model of random loss, the probability distribution of N is specified by:

$$P[N = n] = \begin{cases} (1-q)^n q, & n < N_{max}, \\ (1-q)^{N_{max}}, & n = N_{max}. \end{cases} \quad (25)$$

The window size when the cycle terminates is $W(N, w)$ and the duration of the cycle is $T(N, w)$. The cyclical evolution can now be completely specified as follows. For the mth cycle, let w_m, N_m, and T_m denote, respectively, the window size at the beginning of the congestion avoidance phase, the number of successful transmissions, and the duration. The threshold w_0 for the 0th cycle is assumed to be given. According to our model for random loss, the random variables N_m are independent and identically distributed according to (25). The evolution of w_m and T_m is then specified as follows:

$$w_{m+1} = W(N_m, w_m), \quad (26)$$

$$T_{m+1} = T(N_m, w_m), \quad (27)$$

Equations (25) and (26) specify the transition probabilities for the time-homogeneous Markov chain formed by the $\{w_m\}$. This can be solved to obtain the stationary distribution for the w_m. The long-run throughput is then given by $\bar{\lambda} = \frac{E[N_m]}{E[T_m]}$.

Specifying the deterministic functions $W(n, w)$ and $T(n, w)$ in detail and solving for the stationary distribution is probably more time-consuming than simulation, and is not likely to yield any additional insight. In order to develop an intuitive understanding of how the random loss probability affects the throughput, therefore, we use an approximation for throughput based on the assumption that every cycle begins with a single "average" value $w = w^*$. This yields

$$\bar{\lambda} \approx \sum_n P[N = n] \frac{n}{T(n, w^*)}. \quad (28)$$

The parameter w^* is taken to be the minimum of its value without random loss, $W_{pipe}/2 = (\mu T + B)/2$ (this value is the same for TCP-tahoe and TCP-reno), and a value w_l based only on random loss, and computed in a simple-minded fashion as follows. For a loss probability q, roughly $1/q$ packets are successful in a cycle before a loss occurs. The parameter w is chosen to satisfy the following "fixed point" relationship: given that the parameter for the current cycle is w_l, and that exactly $1/q$ packets are successful, choose w_l such that the window size at the time of loss is $2w_l$, so that the parameter for the next cycle is also w_l. Thus, w_l satisfies

$$W(1/q, w_l)/2 = w_l \quad (29)$$

Since the preceding approximation is fairly drastic, there is no point in solving (29) exactly, so that we feel free to resort to further approximations. Consider TCP-reno first. Starting from $W_0 = w_l$, suppose that the packet loss occurs at time t after the cycle begins. Assuming that the window size does not reach μT (i.e., that random loss has a significant effect in limiting window size), the linear growth in the congestion avoidance phase (see (18)) implies that $2w_l = w_l + t/T$, so that

$$t = T w_l. \quad (30)$$

The number of packets transmitted in time t in the congestion avoidance phase is (see (20)) $(w_l t + t^2/(2T))/T$, so that

$$(w_l t + t^2/(2T))/T = 1/q. \quad (31)$$

Combining (30) and (31), we obtain upon simplification that

$$w_l = \sqrt{\frac{2}{3q}} \quad (32)$$

Starting with a slow start threshold $W_t = w_l$, the evolution for TCP-tahoe is the same as for TCP-reno with $W_0 = w_l$ if one ignores the slow start phase, which is of relatively short duration. The value for w_l obtained by solving (29) for TCP-tahoe is therefore also taken to be given by (32). The way this value is used in subsequent computations is somewhat different, however.

For both TCP-tahoe and TCP-reno, the window size at the beginning of the congestion avoidance phase strongly influences the throughput in the cycle, since the congestion avoidance phase accounts for the bulk of the packets transmitted in the cycle (the slow start phase is relatively short). Thus, in order for the throughput with random loss to be comparable with that without loss, the threshold

$$w^* = min\{(\mu T + B)/2, w_l\} \tag{33}$$

must be comparable (of the same order as) $(\mu T + B)/2$, the threshold in the absence of random loss. From (32), this holds if $q(\mu T+B)^2 = q(\mu T)^2(1+\beta)^2$ is comparable to or smaller than $8/3$. We therefore expect a significant deterioration of throughput for both TCP-tahoe and TCP-reno when $q(\mu T)^2$ is large (say 10 or more). Numerical results, both from the approximate analysis that follows and from simulations, bear out this prediction. Figure 4

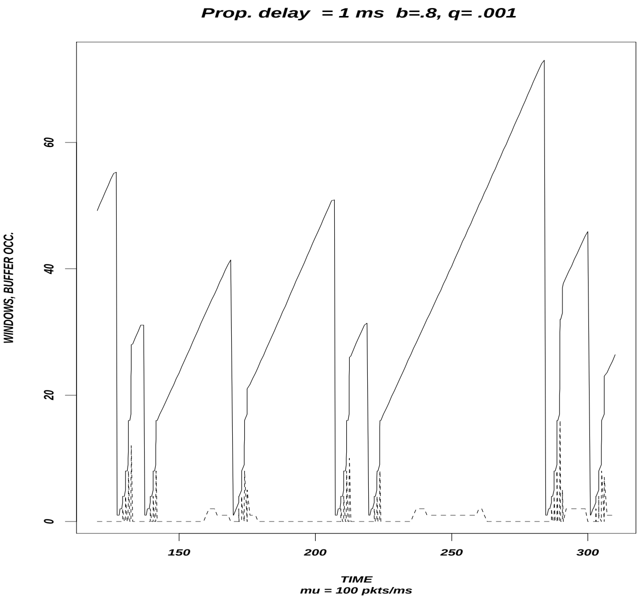

Figure 4: Window and buffer evolution for single connection with high random losses

shows the evolution of window sizes and buffer occupancies with $q = .001$ for the single connection whose lossless evolution was shown in Figure 3. As expected, the slow start thresholds are much smaller than the thresholds for the lossless case (which is 91) and this results in severe throughput deterioration. In the time period shown in the figure, the buffer never overflows and all the packet losses are due to random losses.

Having specified w^*, it remains to compute the approximation (28). Since computation of the deterministic function $T(n, w^*)$ and summation over n is quite tedious, we resort to further simplifications. Let $n(t, w)$ be the number of successful packets at time t, given that the cycle starts at time zero with slow start threshold w. Given w, the function $n(t, w)$ is the inverse of the function of $T(n, w)$, i.e., $t = T(n(t, w), w)$. However, this function is easier to compute, and indeed, has been computed in the previous section, since $\frac{dn}{dt} = \lambda(t) = min\{W(t)/T, \mu\}$.

q	Link Utilization (Analysis/Simulation)			
	$\beta = 0.8$		$\beta = 0.2$	
	Tahoe	Reno	Tahoe	Reno
10^{-1}	.021/.025	.035/.019	.021/.025	.035/.019
10^{-2}	.079/.098	.116/.108	.079/.095	.116/.108
10^{-3}	.291/.343	.369/.381	.291/.340	.369/.379
10^{-4}	.829/.861	.952/.911	.548/.627	.820/.795
10^{-5}	.940/.947	.994/.989	.647/.661	.865/.863
10^{-6}	.952/.953	.996/.994	.659/.656	.870/.870
0	.953/.954	.996/.994	.660/.664	.871/.870

Table 3: Link Utilization as a function of loss probability q for $\mu = 100$, $\tau = 1$, and two values of β ($\beta = 0.8$ and $\beta = 0.2$)

Starting with (28), we change the summation to an integral and then change variables from n to t to obtain

$$\begin{aligned} \bar{\lambda} &\approx \int_n P[N = n] \, \frac{n}{T(n, w^*)} \, dn \\ &= \int_t P[N = n(t, w^*)] \, \frac{n(t, w)}{t} \, \lambda(t) \, dt \end{aligned} \tag{34}$$

The analysis of the previous section can now be directly applied to evaluate this expression. The details are relegated to the appendix. For either TCP-tahoe or TCP-reno, the preceding integral is simple, and we use the general-purpose tool Mathematica to compute it. The result is compared with simulations in Table 3. We consider two values of normalized buffer size for a fixed value of bandwidth-delay product. Although the match in numerical results obtained from approximate analysis and simulation is not as good as in the previous section, the *qualitative* observations are identical. When the loss probability q is relatively high (an order of magnitude or more larger than $(\mu T)^{-2}$), the window size at the beginning of the congestion avoidance phase, and hence the throughput, is dominated by the effect of random losses. This throughput is much smaller than the lossless throughput and is insensitive to the value of β. As the loss probability decreases to $(\mu T)^{-2}$, the throughput gets closer to its lossless value, and the effect of the buffer size on throughput becomes apparent.

The strong dependence of link utilization on $q(\mu T)^2$ is made explicit in Table 4, where we show, for selected values of $q(\mu\tau)^2$ (note that $T = \tau + 1/\mu \approx \tau$ for large bandwidth-delay products), the link utilization for several different bandwidth-delay products. The normalized buffer size β is fixed at 0.8. The dependence on the bandwidth-delay product itself (for fixed $q(\mu\tau)^2$ and fixed β) is weaker, but there is a slight improvement in link utilization as $\mu\tau$ increases. For brevity, we show only the simulation results, but the analytical results exhibit the same qualitative features.

5 Multiple TCP Connections

In this section, we focus on the bias of the TCP window adjustment mechanism against connections with larger round-trip delays. As mentioned in the introduction, this

$q(\mu\tau)^2$	Link Utilization (Tahoe/Reno)		
	$\mu = 100$, $\tau = 1$	$\mu = 200$, $\tau = 1$	$\mu = 200$, $\tau = 2$
10	.343/.381	.363/.398	.377/.406
1	.861/.911	.887/.914	.906/.995
0	.954/.994	.969/.995	.981/.998

Table 4: Link Utilization as a function of $q(\mu\tau)^2$ for three different values of bandwidth-delay product ($\beta = 0.8$)

issue has also been brought up in other studies, e.g., [8], [9]. Our contribution is to provide a more detailed analytical insight into this bias, and to show that this bias is a fundamental property of the TCP window dynamics. Our simulations also reveal the disturbing impact of phase effects on TCP-reno. When simulating TCP-reno, therefore, we will consider a more detailed model of TCP connections going through an ATM switch with randomization in the cell discards when the buffer is full. These simulations serve a two-fold purpose: they enable us to focus on the window dynamics by eliminating the phase effects, and they verify that the analytical results, which are derived for a much simpler model, predict performance well for much more complicated system models.

An intuitive explanation of the bias against connections with longer round-trip times is provided in Section 5.1. Section 5.2 contains a description of the ATM simulation model used for TCP-reno. Section 5.3 contains an approximate analysis of TCP-tahoe and TCP-reno for the simple system model we have considered so far. Numerical results comparing the results of this analysis with simulations are given in Section 5.4.

Throughout this section, we consider N TCP connections. Let $T_i = \tau_i + 1/\mu$ denote the round-trip delay for a packet of the ith connection if it arrives to an empty queue, where τ_i is the propagation delay for the connection. Figure 5 shows the evolution of window sizes for two connections with propagation delays $\tau_1 = 1$ (solid line) and $\tau_2 = 3$ (dashed line). Note that the connection with larger propagation delay operates with a much smaller window size and consequently has a much lower relative throughput. The relative throughputs for the two connections are .817 and .114. For $\tau_1 = 1$ and $\tau_2 = 2$, the corresponding relative throughputs are .628 and .301. We observe empirically that the throughput is inversely proportional to ${\tau_i}^a$, where $1 \leq a \leq 2$.

Another observation from our simulations is that the window evolution for the two connections become synchronized even when the two connections start at different times and despite the fact that the connections have different propagation delays. Synchronized window evolution has been previously reported in [24] for connections with equal propagation delays. The implication of this synchronized evolution is (as pointed out in [24]) that after a period of congestion the sum of connection window sizes may be too small to fill up the available network bandwidth, which results in throughput loss. For tractability, our approximate analysis is based on the assumption of synchronization. It appears very difficult to deduce analytically whether such synchronization would

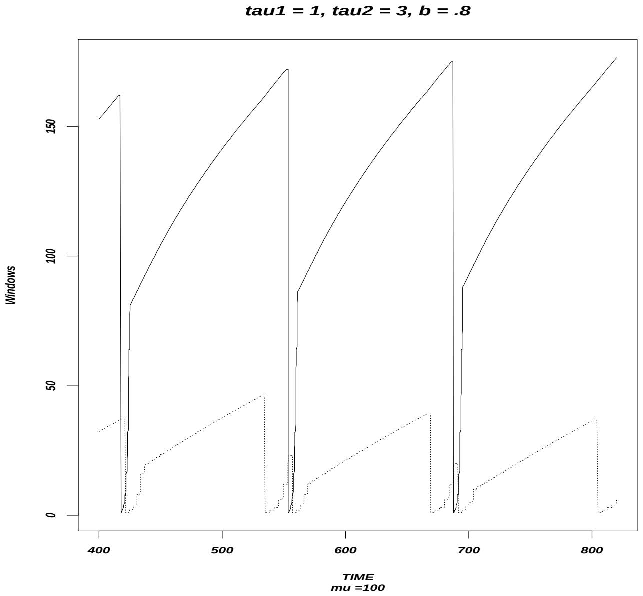

Figure 5: Window evolution for connections with propagation delays 1 and 3

always occur, but the basic observation on which our analysis is based, which is that the window size grows more slowly for connections with higher round trip delays, should apply even if there were instances of evolution without synchronization.

5.1 Intuitive Explanation for the Bias

An approximate expression for the instantaneous throughput for connection i can be obtained using Little's law:

$$\lambda_i = \frac{W_i}{T_i + D_i}, \tag{35}$$

where D_i is the typical waiting time in the queue for a packet from connection i. Note that although D_i is defined as a time-varying quantity, calling it "typical" implicitly assumes that the time variations are slow, as does the application of Little's law, which holds for averaged quantities. Clearly, if all connections had the same window size, and if D_i were small compared to T_i, then the throughput would be inversely proportional to T_i. Moreover, from (16)-(17) in Section 3.2, the growth in window size in the congestion avoidance phase is given by

$$\frac{dW_i}{dt} = \lambda_i / W_i = \frac{1}{T_i + D_i}, \quad 1 \leq i \leq N. \tag{36}$$

Assume now that the connections evolve in synchrony, i.e., that for all connections, the window sizes reach their maximum, and then drop down on detecting packet loss, at roughly the same time. Equation (36) shows that both the throughput and the growth in window size are inversely proportional to T_i (if the waiting times D_i are small). The maximum window size is therefore smaller for connections with larger propagation delays, so that in the next cycle, the initial window size during congestion avoidance is smaller for such connections. Thus, not only are the window sizes smaller throughout the cycle for such connections, but (35) implies that the throughput for such connections would be reduced by a further factor of roughly $1/T_i$. Thus, if there were no queueing delays, the average throughput for connection i could be expected to be inversely proportional to $1/T_i^2$. This unfairness is alleviated somewhat due to all connections suffering roughly equal queueing delays, which accounts for the empirical observation that the throughput is actually inversely proportional to $1/T_i^a$, where $1 \leq a < 2$. Thus, the bias against connections with larger propagation delays is a fundamental consequence of TCP dynamics. As we will see from the numerical results, this is alleviated somewhat when the buffer size is much larger than the bandwidth-delay product, since in that case the round-trip time is dominated by the queueing delay, which at high utilizations is roughly the same for all connections.

5.2 Simulation Model for TCP-reno

TCP-reno differs from TCP-tahoe mainly in its attempt to eliminate the relatively short slow start from the cyclical evolution by means of fast retransmit. Since its window dynamics are otherwise the same as that of TCP-tahoe, we would expect to see the same bias against connections with larger propagation delays in both versions of TCP. However, our simulations for the simple model described in Section 2 reveal the following disturbing feature about TCP-reno: since the number of times the window is halved at the onset of congestion equals the number of lost packets, and since phase effects [9] can cause one connection to systematically lose a larger number of packets, it is possible that a connection gets almost completely shut out. We have observed this effect for two connections with $\tau_1 = 1$ and $\tau_2 = 3$, in which the first connection gets very small throughput even though it has smaller round-trip delay. This effect does not occur in TCP-tahoe because initiation of the slow start phase and the choice of the slow start threshold depends only on at least one packet getting lost, and not on the *number* of packets lost, at the onset of congestion.

In order to eliminate the possibility that phase effects are strictly an artifact of our simple model, we use a detailed simulation of multiple TCP-reno connections over an ATM system. Here, TCP sources are connected to routers with ATM interfaces. We use 576 byte packets which convert to 12 ATM cells at the router output. ACK packets convert to 2 ATM cells. The routers connect to the input ports of an ATM switch over links operating at DS-3 rates. The routers on the sending side assign the right virtual circuit (VCs) for transport through the ATM switch. The VCs in simulated system are configured such that the only bottlenecks are at the ATM switch output ports. The routers at the sending and receiving sides only act as non-bottlenecking hops.

The DS-3 links (using the Physical Layer Convergence Procedure over DS-3) typically operate in a slotted manner with slot duration equal to 10.4 μsecs. However, on the input side to the switch, we do not use slotting on the DS-3 links, so that cell transmissions from the router do not have to wait for a slot boundary. The links connected to the switch output port operate in a slotted manner. The architecture and internal timing of a popular ATM switch are accurately simulated. The switch input ports (where the DS-3 links from the router terminate) are sampled every 2.8 μsecs. A more detailed discussion of internal architecture is not relevant for the performance aspects studied in this paper. It is sufficient to point out that that there is no queueing or blocking internal to the switch, and that contention occurs only at the switch output ports. Congestion is caused by routing many connections to the same output port. The output links from the switch terminate at routers which convert incoming cells to packets and pass the packet to appropriate receivers. Packets with lost cells are dropped at this point.

Mixing slotted and unslotted links is found not to eliminate phase effects. We therefore resort to randomization in buffer discards to eliminate phase effects. With randomizaton, when a cell arrives to a full buffer, instead of dropping the incoming cell, we choose the cell to be dropped randomly from a pool of cells, including the incoming cell and n cells in the tail of the buffer (n is chosen equal to the number of active TCP connections in our simulations).

5.3 Analysis and Numerical Results

We now attempt to develop analytical approximations for the throughput by refining the qualitative explanations given earlier. The analysis is approximate for several reasons: (i) an exact analysis of multiple connections with different propagation delays is not available even for fixed windows, which makes it necessary to approximate the queueing delays D_i in (35) and (36); (ii) it is necessary to further approximate the result of step (i) in order to solve analytically the differential equations arising from a continuous-time approximation to the window evolution; (iii) our assumption of synchronized evolution for all connections does not hold exactly; (iv) our analysis is based on a "fixed point" argument, which assumes periodic behavior that may not hold exactly in the actual system.

In the case of TCP-reno, there is an additional approximation. The number of packets lost by a connection in a congestion episode determines what factor the window size is reduced by. This number varies over different congestion episodes for each connection, and it is difficult to taking this variation into account in the analysis. In the analysis, therefore, we will assume that a constant number of packets are lost by each connection in each congestion episode. Simulation results show that each connection typically loses about two packets on the average in each congestion episode. Losses occur when the buffer is full and one of the connections increases its window by one. When cells from this new packet arrive at the buffer, they typically cause losses to cells belonging to two packets (the tail of the packet arriving from the other connection and the head of the next). So on the average one would expect 3 packets to be lost per collision. In our example of two connections with 40 ms and 80 ms round trip times, once the buffer is full and a collision happens, the congestion episode lasts for 40 ms. During this period the 80ms connection increases its window roughly 50% of the time. So the average number of lost packets due to this increase in 1.5. Therefore 4.5 packets in total, or 2.25 packets per connection, are lost on the average per congestion episode. We will assume in our analysis that each connection loses *exactly* two packets in each congestion episode, so that the new window size is one-fourth of the window size at the onset of congestion.

Note that this assumption is not necessary for TCP-tahoe. As long as each connection loses at least one packet in a congestion episode, all the window sizes drop to one, and the slow start threshold for each connection is set at half of its current window size. Despite the many approximations we make, and despite the fact that the analysis does not account for the details of the ATM simulation model in the case of TCP-reno, we find that the analysis agrees remarkably well with simulations, and thus provides a quick and insightful means of estimating relative throughputs.

Note also that connections evolve in approximate synchrony. This is because whenever a collision happens due to a connection's window increasing when the buffer is full, all cells arriving during the period over which the collision-causing packet arrives are dropped. With the assumption of infinite sources and cell jitters not more than a packet transmission time on an input link, all connections will transmit cells during the time over which the collision-causing packet arrives. Hence, all connections lose packets and halve their windows within the maximum round-trip time.

Assume now that the N TCP-tahoe connections evolve periodically and in synchrony as follows. In the congestion avoidance phase of each cycle, connection i goes from an initial window value of w_i to a value of $2^c w_i$, at which point packets are lost by each connection. Here, $c = 1$ for TCP-tahoe (assuming that each connection loses at least one packet), and $c = 2$ for TCP-reno (assuming that each connection loses exactly two packets). We therefore obtain that the window size at the beginning of the congestion avoidance phase is again w_i (ignoring the slow start phase for the next cycle in the case of TCP-tahoe). We would like to find the value of the w_i such that the preceding "fixed point" behavior holds, and use the resulting window evolution to estimate the throughput obtained by each connection.

In order to solve (36), we must estimate the queueing delays D_i. At any instant of time, the $D_i(t)$ can be taken to be the average delays for an analogous system with *fixed* windows $W_i(t)$. If the link utilization is known to be 100%, and if it is assumed that $D_i \equiv D$ for all i (this has been validated by simulations for fixed windows), then, from (35), the delay D must satisfy the following equation:

$$\sum_{i=1}^{N} \frac{W_i}{T_i + D} = \mu, \qquad \text{for 100\% utilization.} \tag{37}$$

At present, we do not know how to compute the delay when the utilization is *less than* 100 %. In fact, we do not even have a criterion for when the utilization is 100 % (we have derived necessary and sufficient conditions, but these do not coincide). Finally, solving (36) analytically when the D_i are given by the implicit expression (37) appears difficult. We therefore use the following simple approximations. We divide the evolution in the cycle into two phases:

Phase A: $\sum_{i=1}^{N} \frac{W_i}{T_i} < \mu$

Phase B: $\sum_{i=1}^{N} \frac{W_i}{T_i} \geq \mu$.

The condition for Phase B can be shown to be a necessary (but not sufficient) condition for 100 % utilization. If the buffer size is large compared to the bandwidth-delay product, Phase A may not occur at all, since the link may be fully utilized even when the window sizes are reduced following a congestion episode. We must therefore consider two different cases in the analysis.

Case 1: Small buffers

Assume that the link is not always fully utilized during congestion avoidance, and that the queueing delays for all connections are the same, $D_i \equiv D$, where D is given by:

$$D \approx \begin{cases} 0, & \text{in Phase A,} \\ B/(2\mu), & \text{in Phase B,} \end{cases} \tag{38}$$

The motivation for the preceding approximation is as follows. Since the utilization is known to be less than 100 % in Phase A, we assume that there is no waiting in queue in this phase. Phase B, on the other hand, starts from a delay of zero, and ends with a delay of B/μ, assuming that the typical queue length is approximately the same as the maximum queue length towards the end of Phase B (recall that Phase B ends when the latter exceeds B). We therefore use $B/(2\mu)$, the "average" delay value during Phase B, as the value of the queueing delay used in (36) for this phase.

Substituting (38) in (36), we obtain

$$\frac{dW_i}{dt} = \begin{cases} \frac{1}{T_i}, & \text{in Phase A,} \\ \frac{1}{T_i + B/(2\mu)}, & \text{in Phase B,} \end{cases} \tag{39}$$

Starting from an initial value of w_i, we obtain that

$$W_i(t) = w_i + t/T_i, \quad 0 \leq t \leq t_A, \tag{40}$$

$$W_i(t + t_A) = W_i(t_A) + \frac{t}{T_i + B/(2\mu)}, \quad 0 \leq t \leq t_B, \tag{41}$$

where t_A, t_B denote the duration of Phase A and Phase B, respectively. These quantities are obtained as (linear) functions of the w_i as follows. We have

$$\sum_{i=1}^{N} \frac{W_i(t_A)}{T_i} = \mu,$$

which yields, using (40), that

$$t_A = \frac{\mu - \sum_{i=1}^{N} w_i/T_i}{\sum_{i=1}^{N} 1/T_i^2} \tag{42}$$

We now compute the duration of the second phase. This phase lasts until the delay grows to B/μ, at which point the current cycle terminates. Since the link is fully utilized at this time, using (35), we obtain that the duration t_B of this phase is given by the equation

$$\sum_{i=1}^{N} \frac{W_i(t_A + t_B)}{T_i + B/\mu} = \mu \tag{43}$$

Substituting from (40), (42) and (41), we obtain t_B as a linear function of the w_i. Since the window size at the end of the second phase is 2^c times the starting window size for the next cycle, invoking the fixed point condition gives that

$$W_i(t_A + t_B) = 2^c w_i, \quad 1 \leq i \leq N, \tag{44}$$

These are N linear equations in the N unknowns, $w_1, \cdots, w_N$. While we ignored the slow start phase in computing the slow start thresholds w_i, we can now estimate the the duration and number of packets for this period as $t_{ss,i} = T_i log_2 w_i$ and $n_{ss,i} = w_i$, respectively. This yields the following estimate for the average throughput for the ith connection:

$$\bar{\lambda_i} = \{\int_0^{t_A} \frac{W_i(t)}{T_i} dt + \int_0^{t_B} \frac{W_i(t+t_A)}{T_i + B/(2\mu)} dt\}/(t_A + t_B) \quad \textbf{reno}$$

$$\bar{\lambda_i} = \{w_i + \int_0^{t_A} \frac{W_i(t)}{T_i} dt + \int_0^{t_B} \frac{W_i(t+t_A)}{T_i + B/(2\mu)} dt\}/(T_i log_2 w_i + t_A + t_B) \quad \textbf{tahoe}$$

(While the preceding expressions are similar, note that the value of c used in (44) to compute the times t_A and t_B is $c = 1$ for TCP-tahoe and $c = 2$ for TCP-reno.)

Case 2: Large Buffers

The analysis of Case 1 does not apply when the buffer size is large enough that the link is fully utilized throughout congestion avoidance, despite the window size reduction following the onset of congestion. In fact, the analysis in Case 1 yields that t_A is negative, which therefore provides a simple criterion as to when to apply the analysis in Case 2 which is described in the following.

Assuming that the link is always fully utilized during congestion avoidance, let D_{min} be the (unknown) minimum value of the queueing delay seen at the beginning of congestion avoidance. Using the average of this and the maximum queueing delay B/μ to linearly approximate the window evolution described by (36), we obtain that

$$W_i(t) = w_i + \frac{t}{T_i + (D_{min} + B/\mu)/2}, \quad 0 \le t \le t_B \qquad (45)$$

where w_i is the window size of the ith connection at the beginning of congestion avoidance. Using the fixed point argument,

$$W_i(t_B) = 2^c w_i \qquad (46)$$

It follows from (45) and (46) that

$$\frac{w_i}{w_1} = \frac{T_1 + (D_{min} + B/\mu)/2}{T_i + (D_{min} + B/\mu)/2} \qquad (47)$$

Since the link is fully utilized, the following equations must hold:

$$\sum_{i=1}^{N} \frac{w_i}{T_i + D_{min}} = \mu \qquad (48)$$

$$\sum_{i=1}^{N} \frac{2^c w_i}{T_i + B/\mu} = \mu \qquad (49)$$

Using (47)-(49), we can eliminate the w_i and solve for D_{min}. We can substitute back to solve for the w_i and the duration t_B of the congestion avoidance phase. The throughputs are given by

$$\bar{\lambda_i} = \{\int_0^{t_B} \frac{W_i(t)}{T_i + (D_{min} + B/\mu)/2} dt\}/t_B \quad \textbf{TCP}-\textbf{reno}$$

Prop. Delays		Relative Throughputs			
		α_1		α_2	
τ_1	τ_2	Analysis	Simulation	Analysis	Simulation
2	1	.232	.301	.704	.628
3	1	.133	.114	.809	.817

Table 5: Relative throughputs for two TCP-tahoe connections with different propagation delays for $\mu = 100$ and buffer size $B = 80$

Buffer size (cells)	Relative Throughputs			
	α_1		α_2	
	Analysis	Simulation	Analysis	Simulation
256	.1324	.0892	.5285	.5268
2304	.1862	.1385	.6639	.7033
5760	.2539	.2316	.7042	.6966
8064	.2876	.2083	.6918	.7410
14400	.3520	.3388	.6389	.6217
23040	.3941	.3199	.6007	.6318

Table 6: Relative throughputs for two connections with $\tau_1 = 80$ ms, $\tau_2 = 40$ ms, $\mu = 96000$ cells/sec (DS-3 link rate)

$$\bar{\lambda_i} = \{w_i + \int_0^{t_B} \frac{W_i(t)}{T_i + (D_{min} + B/\mu)/2} dt\}/(T_i log_2 w_i + t_B) \quad \textbf{TCP}-\textbf{tahoe}$$

5.4 Numerical Results

For TCP-tahoe, analysis and simulation results are compared for the simple model described in Section 2. Table 5 gives the relative throughputs $\alpha_i = \frac{\bar{\lambda_i}}{\mu}$ for two connections with different propagation delays. Despite the many approximations we have made, the agreement between analysis and simulation is seen to be quite good.

For TCP-reno, simulation results obtained for the ATM system described in Section 5.3 are compared with analytical results for the simple model of Section 2. Table 6 gives the relative throughputs $\alpha_i = \frac{\bar{\lambda_i}}{\mu}$ for two connections with different propagation delays. As described earlier, the analytical results are obtained assuming that exactly 2 losses occur per congestion episode, so that $c = 2$ in (44) and (46). Despite this coarse characterization of the loss behavior at the onset of congestion, and despite all the other simplifying approximations we have made, the agreement between analysis and simulation is quite good for all values of the buffer size.

6 Conclusions

The results stated in the introduction and developed in the rest of the paper imply the following observations (some of which are speculations regarding suitable directions for future research) regarding TCP in high-bandwidth delay product networks.

(1) The use of cumulative acknowledgements in TCP motivates the TCP-tahoe feature of reducing the window size

to one after a loss in order to avoid a burst of packets when the retransmission gets through. This in turn makes an exponential increase of window size in slow start necessary, especially for high bandwidth-delay networks, in order obtain nontrivial link utilizations. On the other hand, this exponential increase causes bursty traffic, which, if the buffer size is smaller than 1/3 of the bandwidth-delay product, causes buffer overflow and a second slow start phase, leading to a lower throughput. TCP-reno tries to avoid this phenomenon by cutting the window to half when it detects a loss. While this does provide better throughput under ideal conditions, TCP-reno in its present form is too vulnerable to phase effects and multiple packet losses to be a replacement for TCP-tahoe. The basic problem with TCP-reno is that there can be multiple window cutbacks due to a single congestion episode, and that multiple losses can lead to a timeout (which in practice can lead to significant throughput loss if coarse timeouts are used). A recently proposed version of TCP (TCP-Vegas) [4] attempts to do this, among other proposed changes, such as more sophisticated processing of round-trip time estimates. A detailed discussion of TCP-Vegas is beyond the scope of this paper. However, in our opinion, in order to significantly improve upon the present versions of TCP, it is necessary to avoid the drastic window reductions in both TCP-tahoe and TCP-reno except when there is sustained congestion (which would cause multiple losses). One possible means of handling isolated losses without changing the window size is by using some form of selective acknowledgement (in which case TCP dynamics should be reworked to take advantage of such an added feature). Pending development of a satisfactory replacement, we recommend the use of TCP-tahoe (together with network level controls to optimize its performance), since it is far more robust than TCP-reno.

(2) TCP's vulnerability to random loss makes it difficult to multiplex data traffic with real-time traffic with a rapidly time-varying rate, especially if both kinds of traffic share the same buffer, as is the case with most current networks. In emerging ATM networks, however, TCP connections might be supported over the UBR or ABR traffic classes, which would typically be buffered separately from higher priority Variable Bit Rate (VBR) or Constant Bit Rate (CBR) traffic. If the latter traffic classes have higher priority, TCP connections would see a time-varying link capacity left over after VBR and CBR traffic has been served. These variations could lead to "random loss", which we have shown can seriously impact performance. However, if there is sufficient buffering to absorb these variations and to keep the end-to-end packet loss probability below the inverse square of the bandwidth-delay product, then our results also imply that TCP performance would not be seriously affected. The results of this paper could therefore form a basis for further investigation as to the buffer sizes required to support TCP over ABR or UBR service classes in ATM. Indeed, we have applied the results here to the design of wireless-wireline interfaces to "hide" the time variations in a wireless channel from TCP/IP [5], so that the random loss seen by TCP is small enough to maintain a high throughput. Again, inclusion of selective acknowledgements in TCP would alleviate this problem, since the window size would not need to be cut back for isolated random losses.

(3) The unfairness of TCP towards connections with higher propagation delays could cause performance problems when multiplexing short and long-haul traffic on WANs. In the short term, these performance problems may not surface because WANs might be significantly underutilized, and there may be no bottlenecks in the path of a TCP connection (i.e., the window size needed for good performance may be too small to cause buffer overflow, so that the maximum steady state throughput is attained). However, for guaranteed performance in highly utilized networks, each TCP connection should be given reserved buffer and bandwidth resources throughout the network. Typically, the resource allocation would be determined at connection set-up and enforced at switches and routers using per connection queueing [6], [21]. Since administering the resources allocated to *every* best effort connection may be excessively expensive, a more feasible alternative might be to allocate and administer resources for an entire traffic class. In such a situation, the unfairness we have pointed out would persist if TCP were supported over the ATM UBR traffic class. However, if TCP is supported over the ABR traffic class, the time-varying rate available to each connection is determined at the network level and is administered at the source, so that different TCP connections should be isolated from each other to a large extent even if they share the same network buffers.

(4) In addition to causing vulnerability to random loss, the fact that loss is the sole means of feedback used by TCP leads to excessive delays. This is because, in networks with high utilizations, the window size for a TCP connection would keep increasing after the bottleneck link is fully utilized, until in fact there is a buffer overflow leading to a loss. The delay and loss performance would improve significantly if we instead used a scheme that tries to maintain a window size which is just large enough to achieve a high link utilization. A scheme such as DECbit [22] attempts to do this using explicit feedback from the switches, and similar schemes are worth pursuing, especially because Explicit Congestion Notification is incorporated as an option for ATM networks. Note, however, that the DECbit scheme in particular shares with TCP the problem of unfairness towards connections with longer propagation delays. Another possibility is a more sophisticated processing of round-trip time estimates similar to the approach taken in [18]. This is certainly attractive, since it avoids the need for explicit feedback. However, if the round-trip delays can change substantially without changes in the load on the path of the connection (e.g., because processing delays at nodes depend on the load on the operating system, or because of delays due to hand-offs for mobile computing applications), then adaptation based on delay processing might be less robust than adaptation based on loss or explicit feedback. In addition, if different connections are not isolated from each other in terms of their use of bandwidth and buffering in the network, then a connection that is more aggressive about obtaining bandwidth by increasing its rate until there is a loss would be at an advantage over connections that process round-trip delays to avoid congestion. Thus, changes at the transport layer must either be adopted universally, or must go hand in hand with network layer controls that guard against greedy connections.

(5) Since the bias against connections with larger round-trip delays is caused by the window adjustment mecha-

nism, it could potentially be removed by modifying the mechanism for probing for bandwidth during congestion avoidance, e.g., by increasing the window size such that the rate of increase of throughput for each connection is the same (such a scheme was considered, but not recommended, in [8]). However, it is not possible to choose a universal time scale for window adjustment which works over networks of different speeds and geographic coverage areas. For instance, probing for extra bandwidth at a rate of 1 Mb/s per second may be too fast for a network with link speeds of 1 Mb/s, but it might be too slow for a gigabit network. Thus, some communication between the network and transport layer would be essential to make such a scheme work. Secondly, such a scheme would still be vulnerable to some of the other drawbacks of TCP, such as performance deterioration with random loss and excessive delays due to probing for bandwidth beyond the point of full link utilization. Thus, this modification may not be the best approach to addressing the problem of unfairness.

In summary, while we have identified several shortcomings of TCP, we have also mentioned possible means of obtaining good performance via network level solutions, such as isolating connections from each other and providing enough buffering to hide excessively fast time variations in available link capacity from TCP. An important topic for future research is, for each context of interest, to translate these into specific recommendations, and to provide systematic design techniques for arriving at these recommendations. Especially interesting is the question of how best to support TCP over the ATM ABR and UBR service class, since that involves adaptation at both the network and transport layers. Another important area for future research is the development of an alternative dynamic window mechanism which addresses some of the shortcomings of TCP while preserving its decentralized nature. Decentralized adaptive rate control while maintaning fairness might be difficult, if not impossible, to achieve, but some enhancements to TCP are undoubtedly possible. Possible improvements might be better congestion avoidance via more sophisticated processing of round-trip delay estimates, and the use of selective acknowledgements to improve the performance in the presence of random loss.

Acknowledgements

We would like to thank Teunis Ott for getting us started on this problem.

Appendix

We give the details of the approximation for the throughput with loss, taking as our starting point equation (34) in Section 4.

We consider only the case $\beta \leq 1$. Considering TCP-reno first, with the threshold at the beginning of the cycle given by $W_0 = w^*$ (given by (33) instead of $W_{max}/2$. Referring to the analysis in Section 3 of the congestion avoidance phase, the instantaneous throughput $\lambda(t)$ is given by:

$$\lambda(t) = \begin{cases} (W_0 + t/T)/T, & 0 \leq t \leq t_A, \\ \mu, & t_A \leq t \leq t_A + t_B, \end{cases} \tag{50}$$

and the number of successful packets by time t is there is no random loss is given by

$$n(t) = \int_0^t \lambda(t)\ dt. \tag{51}$$

Given the simple form of (50), it is easy to compute $n(t)$ explicitly using (51). Substituting into (34), and using (25), we obtain

$$\begin{aligned} \bar{\lambda} \approx & \int_0^{t_A+t_B} q(1-q)^{n(t)}\ \frac{n(t)}{t}\ \lambda(t)\ dt \\ & + (1-q)^{N_{max}}\ \frac{N_{max}}{t_A+t_B}, \qquad \mathbf{TCP-reno} \end{aligned} \tag{52}$$

This is the expression that is plugged into Mathematica to generate the desired results.

For TCP-tahoe, according to our approximation, the threshold for the slow start phase is now given by $W_t = w^*$. Using the case-wise analysis of Section 3.1 for this value of W_t yields t_{ss}, n_{ss} and W_0. We assume for simplicity that there is no random loss in the slow start phase(s), since relatively few packets are transmitted in this phase. As for TCP-reno, the instantaneous throughput in the congestion avoidance phase is given by (50) (it is convenient to take $t = 0$ to correspond to the beginning of congestion avoidance rather than to the beginning of the cycle), and the number of successful packets by time t is given by

$$n(t) = n_{ss} + \int_0^t \lambda(t)\ dt. \tag{53}$$

A simple modification of (52) which includes the duration t_{ss} of the slow start phase yields the following approximation for the average throughput:

$$\begin{aligned} \bar{\lambda} \approx & \int_0^{t_A+t_B} q(1-q)^{n(t)}\ \frac{n(t)}{t+t_{ss}}\ \lambda(t)\ dt \\ & + (1-q)^{N_{max}}\ \frac{N_{max}}{t_{ss}+t_A+t_B}, \quad \mathbf{TCP-tahoe} \end{aligned} \tag{54}$$

References

[1] ATM Forum Traffic Management Specification Version 4.0, Draft Specification ATM Forum/95-0013R11, ATM Forum, March 1996.

[2] J. Bolot and A. U. Shankar, "Dynamical behavior of rate-based flow control mechanisms," *Computer Communication Review*, pp. 35-49, April, 1990.

[3] J. Bolot, "End-to-end packet delay and loss behavior in the Internet," *Proc. ACM SIGCOMM '93.*

[4] L. S. Brakmo, S. W. O'Malley, L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *Proc. ACM Sigcomm,* August 1994.

[5] H. Chaskar, T. V. Lakshman, U. Madhow, "On the Design of Interfaces for TCP/IP over Wireless," *Proceedings of IEEE Milcom '96.*

[6] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proc. ACM SIGCOMM '89.*

[7] K. W. Fendick, D. Mitra, I. Mitrani, M. A. Rodrigues, J. B. Seery, A. Weiss, "An approach to high performance, high speed data networks," *IEEE Communications Magazine,* pp. 74-82, October 1991.

[8] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic," *Computer Communications Review*, vol.21, no.5, pp. 30-47, October 1991.

[9] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," *Internetworking: Research and Experience*, vol.3, no.3, pp.115-156, September 1992. (An earlier version of this paper appeared in *Computer Communication Review*, vol. 21, no. 2, April 1991.)

[10] S. Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance", IEEE/ACM Transactions of Networking, vol. 1, no. 4, pp. 397-413, August 1993.

[11] V. Jacobson, "Congestion avoidance and control," *Proc. ACM SIGCOMM '88*, pp. 314-329.

[12] V. Jacobson, "Modified TCP congestion avoidance algorithm," message to end2end-interest mailing list, April 1990, URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[13] V. Jacobson, "Berkeley TCP evolution from 4.3-tahoe to 4.3-reno," *Proc. of the 18th Internet Engineering Task Force*, Vancouver, August, 1990.

[14] V. Jacobson, R. Braden and D. Borman, "TCP extensions for high performance," RFC (request for comment) 1323, May 1992.

[15] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *ACM Trans. on Computer Systems*, vol. 9, no. 4, pp. 364-373, November 1991.

[16] T. V. Lakshman, U. Madhow, B. Suter, "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance" *Proceedings of IEEE Infocom 1997.*

[17] B. Makrucki, "On the Performance of Submitting Excess Traffic to ATM Networks," *Proc. of Globecom 1991* pp. 281-288, December 1991.

[18] D. Mitra, "Asymptotically optimal design of congestion control for high speed data networks," *IEEE Trans. Commun.*, vol. 40, no. 2, pp. 301-311, February 1992.

[19] D. Mitra and J. B. Seery, "Dynamic adaptive windows for high speed data networks with multiple paths and propagation delays," *Computer Networks and ISDN Systems*, vol. 25, pp. 663-679, 1993.

[20] A. Mukherjee and J. C. Strikwerda, "Analysis of dynamic congestion control protocols – a Fokker-Planck approximation," *Proc. ACM SIGCOMM '91*, pp. 159-169.

[21] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks–the multiple node case," *Proc. IEEE Infocom '93.*

[22] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer," *Proc. ACM SIGCOMM '88*, pp. 303-313.

[23] S. Shenker, "A theoretical analysis of feedback flow control," *Proc. ACM SIGCOMM '90*, pp. 156-165.

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

[25] G. R. Wright and W. R. Stevens, "TCP/IP Illustrated, Volume 2, The Implementation", Addison Wesley, 1995.

[26] L. Zhang, "A new architecture for packet switching network protocols," Ph. D. dissertation, M.I.T. Lab. Comput. Sci., Cambridge, MA, 1989.

[27] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic," *Proc. ACM SIGCOMM '91*, pp. 133-147.