The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss

T. V. Lakshman, *Member, IEEE,* and Upamanyu Madhow, *Senior Member, IEEE*

***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) packets may incur random loss (e.g., due to transient congestion caused by fluctuations in real-time traffic, or wireless links in the path of the connection). 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. Second, for multiple connections sharing a bottleneck link, TCP is grossly unfair toward 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, 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.**

***Index Terms*—Flow control, congestion control, error recovery, Internet, TCP/IP, transport protocols.**

I. 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.[1] This assumption may not hold for the wide-area networks (WANs) formed by the interconnection of LANs using high-speed backbone networks. 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 impact of these changes on the performance of the most popular data transfer protocol in current use, TCP/IP. 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 LANs, 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). 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 networks with both wireless and wireline links, losses and time variations due to wireless links in the path of the connection can also be accommodated 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 VI, the results here should provide a basis for further work on developing network level design guidelines for supporting TCP.

One of the simplifications of the model used for our analysis is that two-way traffic (and the accompanying *ack compression*

Manuscript received June 20, 1995 revised February 25, 1997; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Mitra. This work was supported in part by the U.S. Army Research Office under Grant DAAH04-95-1-0246.

T. V. Lakshman is with the High Speed Networks Research Dept., Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: lakshman@research.bell-labs.com).

U. Madhow is with the ECE Department and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801 USA (e-mail: madhow@uiuc.edu).

Publisher Item Identifier S 1063-6692(97)04489-0.

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

[27]) 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 V).

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

1) While TCP-reno produces less bursty traffic than TCP-tahoe, it is much less robust toward "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 toward 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]. Our model is similar to that used in [24], but the key differences between our paper and previous studies are that: 1) the ratio of bandwidth-delay product to buffer size is much higher in our study and 2) 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 earlier studies. Furthermore, in contrast to previous studies, we place more emphasis on detailed analytical insight on the effects of various parameters on performance.

The bias of TCP-tahoe against connections with large round-trip delays and against connections traversing a large number of congested gateways has been noticed in other studies of TCP-tahoe [8], [9], [26]. A heuristic analysis in [8] shows that, for multiple connections sharing a bottleneck link, the round-trip time. While we consider a similar system in Section V, our analysis is more detailed, taking explicit account of the buffer size and the bandwidth-delay product. Oscillatory behavior and unfairness toward connections with larger propagation delays have also been noticed in a previous analytical study of feedback-based congestion control [2]. Other analyses of flow control schemes include [20], [22], [23], but these references do not address the specific concerns raised here in any detail.

The system model is described in Section II. Analytical and simulation results for the evolution of a single connection in the absence of random loss are given in Section III. Section IV considers the effect of random loss. Section V contains results for multiple connections with and without random loss. We give our conclusions in Section VI.

II. System Model

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. 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: 1) the time between the release of a packet from the source and its arrival into the link buffer; 2) the time between the transmission of the packet on the bottleneck link and its arrival at its destination; and 3) the time between the arrival of the packet at the destination and the arrival of the corresponding acknowledgment at the source. The propagation delay for a packet from the ith connection is denoted by τ_i.

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 acknowledgment 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.

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

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 acknowledgment (i.e., ignoring acknowledgment for retransmitted segments) and computing a weighted average of the old and new estimates. Refer to [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 ms. 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 acknowledgments* (i.e., multiple acknowledgment 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.

A. 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 acknowledgment 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 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 III, 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 IV, on the other hand, where we consider random loss, the duration of, and window evolution within, different cycles is random.

B. Description of TCP-reno

After the number of duplicate acknowledgments exceeds a threshold (typically three), TCP-reno retransmits the packet. However, instead of cutting the window back to one, it only reduces it by a factor of two. Further, 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 acknowledgment until the "next expected" number in the acknowledgment 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 nonrepeated acknowledgment, 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 acknowledgment 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 acknowledgment. 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

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].

III. Evolution Without Random Loss

We consider the evolution of a single connection and derive expressions for its long-term throughput. Define the normalized buffer size $\beta = B/(\mu\tau + 1) = B/\mu T$, 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

$$W_{\text{pipe}} = \mu T + B = \mu\tau + B + 1. \tag{1}$$

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 $W_t = W_{\text{pipe}}/2$, followed by congestion avoidance until $W = W_{\text{pipe}}$. 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 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 $W = W_t = W_{\text{pipe}}/2$, does congestion avoidance until $W = W_{\text{pipe}}$, and drops back to $W = W_t = W_{\text{pipe}}/2$ after a packet loss due to buffer overflow is detected via duplicate acknowledgment.

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 $\overline{\lambda} = N_c/T_c$. In the following, we describe this evolution more carefully, and compute these quantities in sufficient detail to produce an excellent match with simulations.

TABLE I
Evolution During Slow Start Phase

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
⋮				

A. 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 acknowledgment in this phase, so that two packets are released into the buffer for every acknowledgment. Table I shows the evolution of the window size and the queue length in this phase. For every acknowledgment, 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 I 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 acknowledgment 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, acknowledgment 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 acknowledgment, 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,

$$W(nT + m/\mu) = 2^n + m + 1, \qquad 0 \leq m \leq 2^n - 1 \tag{2}$$

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

$$Q(nT + m/\mu) = m + 2, \qquad 0 \leq m \leq 2^n - 1. \tag{3}$$

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

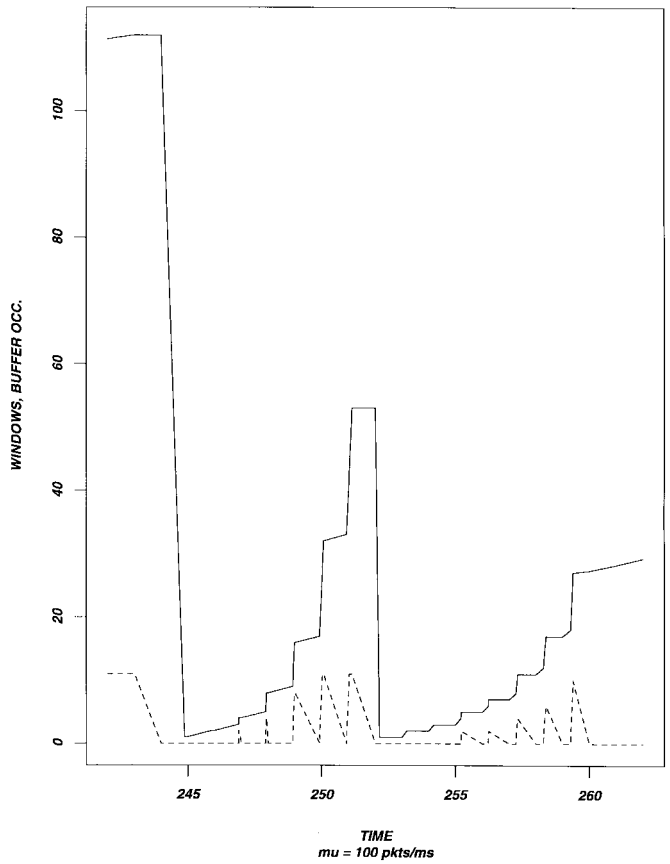

Fig. 1. Window and buffer evolution for a single connection: Two slow starts. Prop. delay = 1 ms; b =0.1.

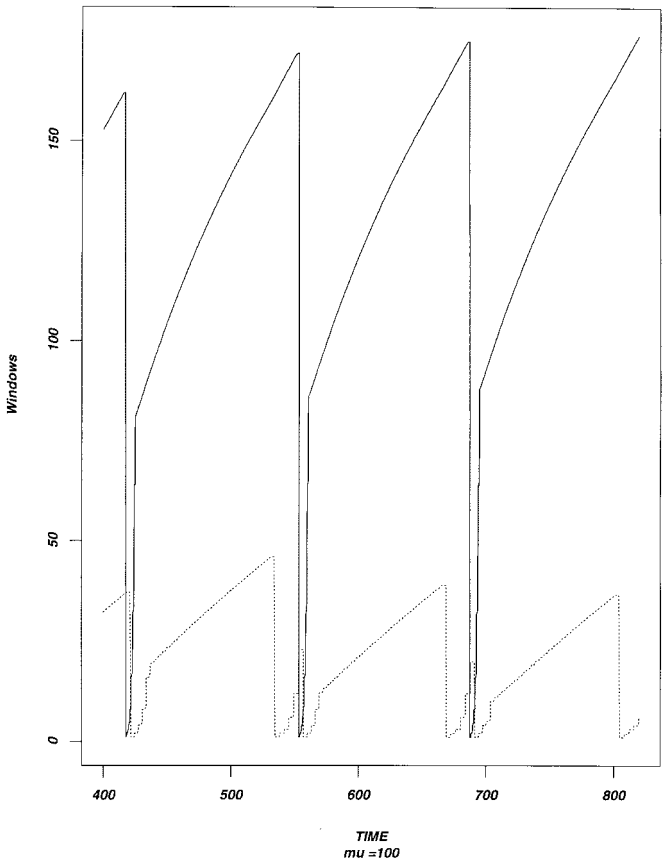

Fig. 2. Window and buffer evolution for a single connection: one slow start. $tau1 = 1, tau2 = 3, b = 0.8$.

size as follows. Define the integer $n_b = \lceil \log_2 (B-1) \rceil$, so that $2^{n_b-1}+1 < B \leq 2^{n_b}+1$. 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

$$W_b = 2^{n_b} + B. \tag{4}$$

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

$$W_b \leq W_t. \tag{5}$$

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 $W_{\text{pipe}} = \mu T + B$, the slow start threshold equals $W_t = W_{\text{pipe}}/2 = (1+\beta)\mu T/2$. Since $W_b \approx 2B = 2\beta\mu T$, the condition for buffer overflow (5) is approximately equivalent to $\beta \leq 1/3$.

Fig. 1 shows the simulated congestion window and buffer occupancy evolution for a single connection using TCP-tahoe with $\tau = 1$, $\mu = 100$, and $\beta = 0.1$ (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 $W_{\text{pipe}} = \mu\tau + B + 1 = 111$ 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

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 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 $W_{\text{pipe}} = 111$, after which a new cycle begins.

When β is greater than 1/3 the window evolution for TCP-tahoe is different. This is illustrated in Fig. 2, which shows the evolution of window sizes and buffer occupancy for $\beta = 0.8$. Here, packet loss is seen to occur when the window is of size $W_{\text{pipe}} = 181$. 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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.