TCP/IP Performance with Random Loss and Bidirectional Congestion

T. V. Lakshman[†] U. Madhow[*] Bernhard Suter[†]

† Bell Labs
101 Crawfords Corner Road
Holmdel, NJ 07733
USA

∗ University of Illinois
Electrical and Computer Engineering Department
Urbana, Illinois 61801.
USA

Abstract

With the envisaged growth in Internet access services over networks with asymmetric links such as Asymmetric Digital Subscriber Line (ADSL) and Hybrid Fiber Coax (HFC), it becomes crucial to evaluate the performance of data transport protocols over systems in which the bottleneck link speed on the reverse path (i.e., the path followed by acknowledgements) is considerably slower than that on the forward path (i.e., the path followed by data packets). In this paper, we provide guidelines for designing network control mechanisms for supporting TCP/IP, the widely used Internet transport protocol, over such asymmetric networks. The key results underlying these guidelines are as follows:

(1) For both the prevalent Tahoe version with Fast Retransmit and the Reno version of TCP, we determine the throughput as a function of buffering, round-trip times and *normalized asymmetry* (defined as the ratio of the transmission time of acknowledgements (ACKs) in the reverse path to that of data packets in the forward path). We identify three modes of operation which are dependent on the forward buffer size and the normalized asymmetry, and determine the conditions under which the forward link is fully utilized. We also show that drop from front queueing of ACKs on the reverse link provides performance advantages over FIFO queueing.

(2) Asymmetry increases TCP's already high sensitivity to random packet losses that occur on a time scale faster than the connection round trip time (e.g., caused by transient bursts in real-time traffic). We generalize the by now well-known relation relating the square root of the random loss probability to obtained TCP throughput, originally derived considering only data path congestion. Specifically, random loss leads to significant throughput deterioration when the product of the loss probability, the *normalized asymmetry* and the *square* of the bandwidth delay product is large.

(3) Congestion in the reverse path adds considerably to TCP's unfairness when multiple connections share the reverse bottleneck link. We show how such problems can be alleviated by per connection buffer and bandwidth allocation on the reverse path.

1 Introduction

It is expected that high-speed Internet access services to residential subscribers will be provided using asymmetric access networks, such as networks using ADSL and its variants, HFC cable networks, and combinations such as a downstream path (network to subscriber) over a cable or satellite link and a telephone upstream link (subscriber to network or service provider). These systems have an inherent bandwidth asymmetry which could be as low as 10 for some of the proposed cable modem and ADSL access services, or as high as 100 or more when a telephone return path is used. Even higher asymmetries in the bandwidths seen by downstream connections will result if the access traffic is bidirectional, hence causing the slow upstream link from the subscriber to the network to be shared by both data packets (where the destination is upstream) and ACK packets (where the destination is downstream). Indeed, data traffic from other connections can cause congestion in the acknowledgement path of a given connection even when there is no raw bandwidth asymmetry.

Many data applications over asymmetric networks (such as Web browsing or file transfers) are built on TCP/IP [13], the widely used Internet data transport protocol. Our goal is to obtain a detailed understanding of the performance of TCP over asymmetric networks, with a view to providing system design guidelines for supporting TCP over such settings.

We consider TCP connections that use a fast *forward path* for data packets, and a slow *reverse path* for ACK packets, which arise naturally from applications that truly exploit asymmetric networks (e.g., Web browsing from a home computer). Since TCP uses the arrival rate of ACKs to control data packet flow, it is important to determine whether congestion in the ACK path leads to poor utilization of network resources for such applications. We will refer to connections of the type just described as *forward connections,* and obtain basic analytical insight (verified using simulations) into the effects of asymmetry by studying the throughput performance of such connections. We also consider *reverse connections,* which generate data traffic on the reverse path and ACK packets on the forward path, since many applications (e.g., file transfers from home to work) would require such connections. Note that, in the context of asymmetric networks, it is not of interest to consider reverse connections on their own, since earlier studies of TCP/IP that ignore ACK path congestion would apply in this case [7], [12], [14], [15]. However, we show via simulations that the interaction between forward and reverse connections has serious performance implications, give qualitative explanations for the observed phenomena, and suggest methods for bandwidth sharing between forward and reverse connections.

We study both the prevalent Tahoe version of TCP with fast retransmit and the later Reno version [3, 4, 13] (henceforth referred to as TCP-Tahoe and TCP-Reno, respectively). While it has been observed that many Internet connections are of short durations, the impact of a slow acknowledgement channel on TCP congestion avoidance behavior is best understood by considering persistent sources, which could be thought of as modeling long file transfers. We therefore restrict attention to the latter throughout this paper.

In addition to the *raw asymmetry* in the forward and reverse link speeds (i.e., the ratio of the forward bottleneck link speed to the reverse bottleneck link speed, where both are expressed in bits/second), it is useful to define the *normalized asymmetry* as the ratio of the transmission time of ACKs on the bottleneck link on the reverse path to that of packets on the bottleneck link on the forward path (i.e., the normalized asymmetry is the ratio of the forward bottleneck link speed in packets/second to the reverse bottleneck link speed in ACKs/second). Note that the normalized asymmetry is typically much smaller than the raw asymmetry, since TCP ACKs (typically about 40 bytes) are much shorter than data packets (typically 500 to 1000 bytes), and can be shortened further using header compression. It will turn out that, when there are only forward connections in the system, the parameter directly affecting TCP performance is the normalized asymmetry rather than the raw asymmetry, which alleviates the impact of the asymmetry. However, we will see that, when the reverse link is shared by both forward and reverse connections, the raw asymmetry plays a significant role.

The remainder of this section contains a brief account of our main results and how they fit in with the existing literature on TCP performance. In order to obtain basic insight into the effect of asymmetry, we consider an idealized model consisting of a single forward and reverse link, together with a propagation delay. For our most basic results, we use a combination of approximate analysis and simulations to characterize the behavior of a single forward connection as a function of the normalized asymmetry, the buffering on the forward and reverse links, and the bandwidth-delay product. Depending on the forward buffer size and the normalized asymmetry, there are three distinct modes of operation. It is shown that the forward buffer size must be at least as large as the normalized asymmetry in order for the most desirable mode (i.e., the one in which the forward link is fully utilized) to be in operation. It is also shown that drop from front queueing of ACKs on the reverse link provides performance advantages (significant in some regimes) over FIFO queueing.

Next, we consider a situation in which packets may be lost randomly in the forward path. Random loss for our purposes is not necessarily due to link errors. Rather, it includes all losses that occur on a time scale faster than the round trip delay of the connection (e.g., transient congestion caused by high priority real-time cross traffic). While an ideal congestion control mechanism should react only to loss due to sustained congestion (i.e., loss due to mechanisms that persist over one or more round trip times), TCP reacts to all losses by scaling back its transmission rate. Thus, too large a level of random loss leads to a significant reduction in TCP throughput. A setting of particular interest is TCP supported over an Available Bit Rate (ABR) ATM virtual circuit, where transient congestion in the ATM network may cause TCP packet loss. Finally, random loss may also occur in heterogeneous networks which contain lossy wireless links of time-varying quality.

The effect of these transient or random losses on TCP throughput is well studied, and it has been shown that, assuming that there is congestion only on the data path, the obtained TCP throughput is inversely proportional to the product of the round trip time and the square root of the loss probability [7], [8], [6], [10]. We show here that bidirectional congestion increases TCP's sensitivity to loss. In particular, for asymmetric networks, the obtained TCP throughput is inversely proportional to the product of the round trip time, the square root of the loss probability, *and* the square root of the normalized asymmetry. Note that the original loss-throughput formula in [7], [8], [6], [10], has been used to define the notion of *TCP-friendliness* for identical losses, a flow whose throughput does not reduce as much as expected by the TCP loss-throughput relation is considered TCP unfriendly. Because of the possibility that TCP-unfriendly flows will starve TCP (and TCP-friendly sources) flows when sharing network resources, it has been advocated that all applications should use TCP-friendly flow control. Our result shows that, for networks with significant asymmetry, the notion of TCP-friendliness would have to be modified.

For multiple TCP connections sharing a forward bottleck link, it has been shown in several previous studies [1], [2], [8] that, assuming no congestion on the ACK path, connections with larger round trip times get a smaller share of the bandwidth. In this paper, we present simulation studies that show that, for multiple forward connections, ACK path congestion exacerbates this inherent unfairness of TCP in the absence of network level control of bandwidth and buffer allocation on the reverse link. In particular, First-In-First-Out (FIFO) sharing of the reverse link by ACKs for different forward connections can lead to *lockout* periods of unpredictable, and long, durations when several connections have essentially zero throughput. Furthermore, when (ACKs for) forward connections share the reverse link with (data packets for) reverse connections, FIFO sharing of the reverse link leads to a nearly complete lockout of the reverse connections. We show that the preceding performance problems can be alleviated by using fair queueing (or weighted round robin with per connection queueing) on the reverse link. However, even with carefully implemented fair queueing on the reverse link, large disparities in size between data packets for reverse connections and ACKs for forward connections can lead to the phenomenon of *ack starvation* on the reverse link. [1] We show that such ack starvation can be particularly problematic for asymmetric networks, and discuss methods for addressing this problem.

Previous simulation studies of TCP-Tahoe include [14], [12], [15]. Simulations for the simple multi-hop network considered in [14] showed the oscillations in window sizes and the unfairness of TCP towards connections traversing a larger number of hops. In [12], 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. An analytical and simulation study of TCP (Tahoe and Reno) with no reverse path congestion is presented in [7], [8]. The effect of different buffer sizes relative to the bandwidth-delay product and the effect of random losses on throughput is derived. The effects of two way traffic are considered in [15], where the phenomenon of *ack compression* resulting from the queueing of acknowledgements is pointed out. However, none of these previous studies have studied the scenario analyzed in this paper, viz., TCP performance with a slow ACK channel where the primary bottleneck is in the reverse path.

The system model and relevant aspects of TCP are described in Section 2. Analytical and simulation results for the evolution of a single connection (including the effect of random loss) are given in Section 3. Section 4 contains simulation results for multiple connections. Concluding remarks, including a summary of design guidelines, are in Section 5.

2 System Model

2.1 Forward and Reverse Path Model

Our model is similar to that considered in [12], [15], [7], with the following key difference: the path followed by acknowledgements is explicitly modeled. We consider an infinite data source which always has packets to send, so that the units of data on the forward path are maximum sized packets. We consider a forward link with capacity μ_f packets per second and a FIFO forward buffer of size B_f packets. For each packet that is received by the destination, a cumulative acknowledgement is generated which contains the next expected segment number. The fact that acknowledgements are cumulative implies that later ACKs contain at least as much information about which packets have reached the destination as earlier ACKs, so that ACK loss simply results in bursty traffic on the forward path. This is the key to our analysis of the impact of asymmetry on performance. It is assumed that the speed of the reverse link is μ_r ACK packets per second, and the the reverse buffer is FIFO of size B_r ACK packets. The forward and reverse buffer sizes do not include the packet in service. We assume that $\mu_f = k\mu_r$, with $k > 1$, where k is the *normalized asymmetry factor*, i.e. the effective bandwidth asymmetry normalized by the ratio of forward to reverse packet sizes..

While the path traversed by the connection may include links other than the forward and reverse links explicitly modeled here, it is assumed that the performance is dominated by the bottleneck

[1]The term *ack compression* was coined in [15] for the phenomenon of multiple acks arriving in quick succession after being queued behind a data packet. We use the term ack starvation here because of the large number of acks lost during the time that a data packet is served on the slow reverse link.

links in each direction. The remainder of the network is modeled via a constant "propagation" delay, which includes queueing delays on other links, propagation delays, and processing delays at nodes in the path of the connection. The propagation delay on the forward path is defined as the time between when a packet is completely transmitted on the forward link and when it arrives at the destination, and is denoted by τ_f. The delay on the reverse path is analogously defined as the time between when an ACK packet completes transmission on the reverse link and when it arrives at the source, and is denoted by τ_r. While we distinguish these two delays in our notation because of the asymmetry between the forward and reverse paths, we will see that only the net propagation delay $\tau = \tau_f + \tau_r$ affects performance. This delay can range from several milliseconds to several hundreds of milliseconds. The *round-trip time* is the time between when a packet is sent by the source and when the source receives an acknowledgement for that packet, and includes queueing at the forward and reverse links in addition to the propagation delay. The average round-trip time will be denoted by T.

We model random loss as follows: each head-of-the-line packet in the forward buffer has a probability q of being lost after transmission, and different packet losses are independent. While much of the analysis will be for the case $q = 0$, we will show that random loss can have a significant impact on performance.

The model described above is shown in Figure 1.

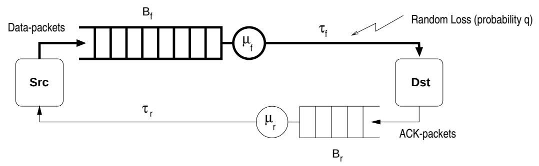

Figure 1: System model for a single connection

The preceding model is for a connection that uses the fast forward link for data packets and the slow reverse link for ACK packets. Henceforth, we will call such connections *forward* connections. We will only consider forward connections in Section 3, since the goal there is to determine whether TCP is able to utilize the fast forward link satisfactorily. In Section 4, we also introduce *reverse* connections, which use the slow reverse link for data packets, and the forward link for ACK packets. Our objective is to determine under what circumstances both forward and reverse connections obtain satisfactory link utilizations.

Throughout the paper, it is convenient to consider the following example.

Running Example:

The reverse link is of speed 320 Kbps, ACK packets are of length 40 bytes. Forward packets are of length 1000 bytes. The reverse link speed μ_r is therefore 1000 ACKs/s, so that a asymmetry factor $k = 5$ (i.e., a μ_f of 5000 packets/s) corresponds to a raw forward link speed of 40 Mb/s. [2] The propagation delay τ for the simulations shown in Section 3 to aid the analytical development is kept fixed at 2 milliseconds, although it is varied in our numerical results in Section 4 in order to illustrate several points. The forward buffer size $B_f = 8$ data packets. The reverse buffer size is $B_r = 5$ ACK packets.

[2]In practice, both the forward and reverse link speeds might be scaled down considerably in order to support multiple connections, but normalized asymmetry factors of up to 25 could occur.

2.2 Background on TCP

The connection of interest uses a window flow control protocol. At time t, the window size is denoted by $W(t)$, and is equal to the maximum allowed number of unacknowledged packets (not counting retransmissions). Since we assume an infinite data source, the connection uses its allowable window to the fullest extent, i.e., at time t, there are indeed $W(t)$ unacknowledged packets. The window varies dynamically in response to acknowledgements and to detection of packet loss. Upon receiving a packet, the destination is assumed to send a cumulative acknowledgment back immediately. Even though selective ACKs or NACKs are not available, a single packet loss can be detected by consecutive acknowledgements having the same "next expected" number. Both TCP-Tahoe and TCP-Reno therefore have a *fast retransmit* option, in which a packet is retransmitted after the number of such *duplicate acknowledgements* exceeds a threshold (typically three). If packet loss is not detected in this manner, it leads to expiry of a timer. In either case, TCP-Tahoe drops its window to one upon loss detection. Subsequently, the window grows rapidly, by one packet for every successfully acknowledged packet, until it reaches half of the window size at the last packet loss. This (typically short) stage of rapid window growth is paradoxically called *slow start*, since it is slow compared to not having decreased the window at all after a loss. After slow start, the algorithm switches to *congestion avoidance,* in which the window grows slowly in order to probe for extra bandwidth, by incrementing the window size by one for every window's worth of acknowledged packets. This growth continues until the maximum window size is reached, or until another packet loss is detected.

TCP-Reno is similar to TCP-Tahoe, except that it tries to avoid the slow start phase by remaining in congestion avoidance unless there is a timer expiry. Packet loss detected via duplicate ACKS results in the window being cut by half. If a timer expiry does occur, then the window size is dropped to one, and slow start is used to grow the window back to half its value when the timer expired.

The operation of the timer is as follows. When a packet is sent, a timeout value is computed and a 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. In principle, the timer value is determined using estimates of the mean and variance of the round-trip time [5]. However, the granularity of the timer used in most practical implementations is usually much coarser than the estimates. In our results, we consider a coarse-grained timer with a granularity of 500 milliseconds (i.e., the time between the release of a packet and the expiry of the timer associated with it is at least 500 milliseconds).

See [13] for a detailed description of TCP, [3] for the original version of TCP-Tahoe, and [4] for a description of TCP-Reno. The following description of the window evolution is sufficient for our purpose.

Description of TCP-Tahoe and TCP-Reno

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

(1 Tahoe/Reno) After every non-repeated ack:
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 Tahoe) After a packet loss is detected (when timer expires or the number of re-

peated ACKs exceeds a threshold):
set $W_t = W/2$;
set $W = 1$.
(2 Reno) When the number of repeated ACKs 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 Reno) Upon timer expiry, the algorithm goes into slow start as before:
set $W_t = W/2$;
set $W = 1$.

In addition to the preceding steps, TCP-Reno incorporates the following refinement: while it cuts its window by half after detection of each packet loss via duplicate ACKs, in order to prevent a burst of packets from being transmitted when the retransmission is finally acknowledged, it temporarily expands the window size to permit new packets to be transmitted with duplicate ACKs until the "next expected" number in the acknowledgement advances. The window expansion takes place after $W/2$ ACKS have been received, in order to ensure that the number of outstanding packets has been reduced to the new window size.

3 A single forward connection

We provide approximate analysis that yields insight into TCP evolution, and explains the dependence of performance on various system parameters. Section 3.1 provides basic approximations for a fixed congestion window W which we will use in our analysis. These include a modification of Little's law for an asymmetric system. Sections 3.2 and 3.3 provide performance analyses for TCP-Tahoe and TCP-Reno, respectively, for a single connection with no random loss. It is shown that TCP-Reno, while supposedly an improved version of TCP-Tahoe, actually performs worse in this setting. Section 3.4 considers the performance of TCP-Tahoe and TCP-Reno in the presence of random loss, and shows that the throughput exhibits a thresholding effect.

In order to motivate the detailed calculations in Sections 3.2 and 3.3, the results in these sections are summarized in Figure 2, which shows the throughput of TCP-Tahoe and TCP-Reno as a function of the normalized asymmetry. The system considered is the running example described in Section 2, with a propagation delay of 2 ms. The match between analysis and simulation is excellent, which implies that the analytical description of TCP behavior in Sections 3.2 and 3.3 is reasonably accurate within the considered framework. The figure shows that there are several different regimes of operation. These are determined by the relationship between the normalized asymmetry and the forward buffer size. Further discussion of these results will be given in the following.

3.1 Analysis for Fixed Windows

Assume that the connection is using a fixed window size of W, which can be thought of as a snapshot of a TCP window at a given time. The minimum round-trip time T incurred by a packet is given by $T = \tau + \frac{1}{\mu_r} + \frac{1}{\mu_f}$. This subsection is based on applying Little's law (and a modification thereof) as

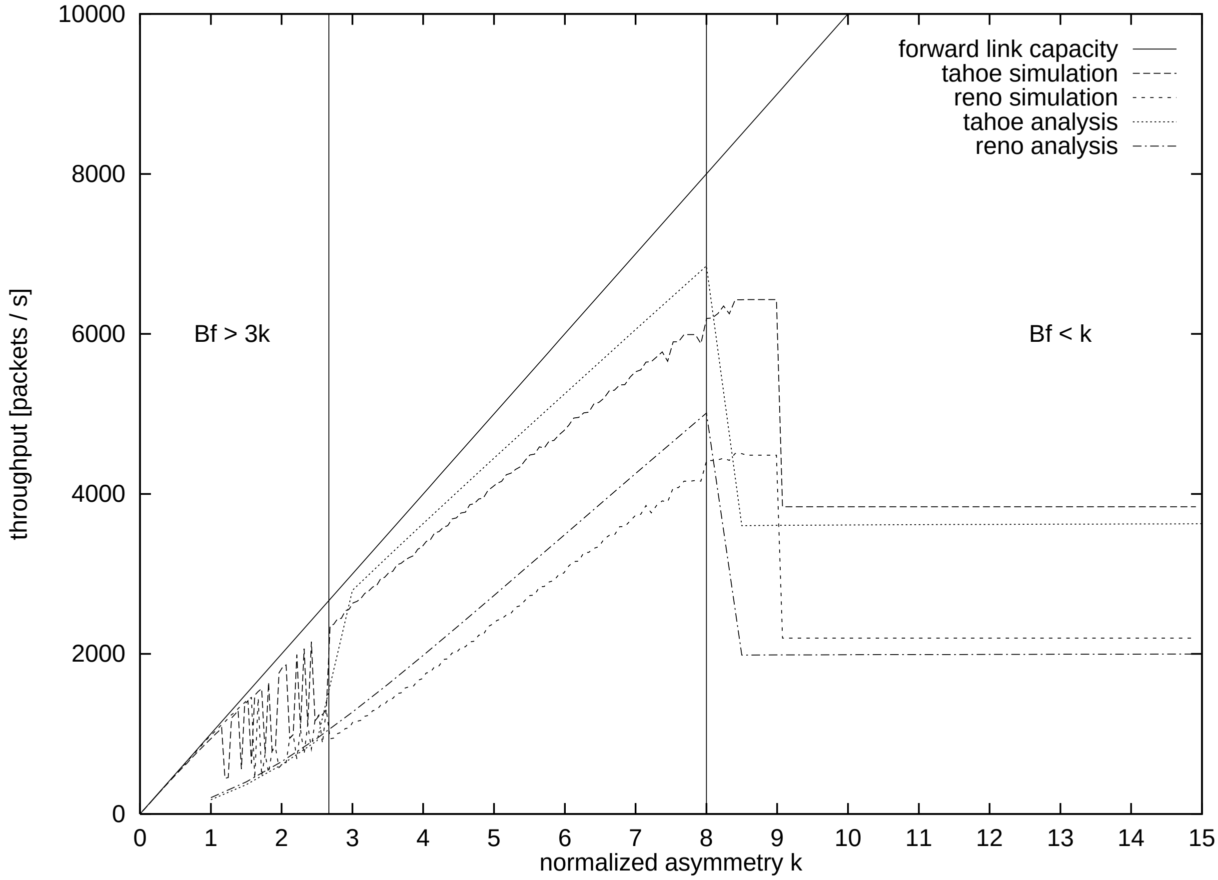

Figure 2: Analytical and simulation results of the throughput of TCP-Tahoe and TCP-Reno as a function of the normalized asymmetry k.

a means of relating window size (or the number of outstanding packets) to the throughput (or the rate of successful transmission of forward link packets). The analysis gives optimistic throughput estimates, since it assumes that the minimum possible average delay is incurred by a packet. This assumption of traffic that is as smooth as possible is made for analytical convenience, but is a good approximation for the deterministic system model in Section 2. Refinements to the analysis will be introduced as necessary when explaining the behavior with dynamic windows.

Condition for fully utilized reverse link

Applying Little's law, we get the following condition for the reverse link to be fully utilized: $W \geq \mu_r T$. This condition is necessary but not sufficient, since the round-trip delay incurred by a packet can be larger than the minimum round-trip time T due to bursty arrivals at the forward or reverse buffer.

Condition for attaining throughput $\lambda > \mu_r$

If the rate of packet transmissions on the forward link is λ, ACKs arrive at the reverse link at the same rate. If $\lambda > \mu_r$, on average only one out of every λ/μ_r ACKs get through on the reverse link, the others have to be dropped as the reverse link buffer is overflowing. Thus, on average, each ACK in the reverse buffer and each ACK in flight in the reverse link actually represents acknowledgement information for λ/μ_r forward packets. [3] Since the reverse pipe is completely full, any ACK that does make it into the reverse buffer sees B_r ACKs ahead of it, so that the net delay it incurs in the reverse buffer (including its service time) is B_r/μ_r.

The modified Little's law that results from the preceding arguments reads as follows:

$$W = \lambda(\tau_f + \frac{1}{\mu_f}) + \frac{\lambda}{\mu_r}[\mu_r(\tau_r + \frac{1}{\mu_r}) + B_r] \tag{1}$$

The first term corresponds to the number of packets in the forward pipe, while the second is the number of surviving ACKs in the reverse pipe, multiplied by λ/μ_r, the number of forward packets represented by each surviving ACK on average. This yields the following estimate of the throughput λ as a function of the window size.

$$\lambda = \frac{W}{T + \frac{B_r}{\mu_r}} \tag{2}$$

The preceding argument assumes that the forward buffer is large enough to withstand the bursty traffic resulting from ACK loss. A necessary condition for this is that the forward buffer is larger than the average burst size, i.e., that $B_f \geq \frac{\lambda}{\mu_r}$. This condition is not sufficient: loss on the forward path can occur even when this condition is satisfied, since not all bursts contain an equal number of packets.

The window size W_{full} for which the forward link can be fully utilized is obtained by setting $\lambda = \mu_f$ in (1), which yields $W_{full} = \mu_f(T + B_r/\mu_r) = \mu_f T + kB_r$. For the running example, $W_{full} = 41$ for an asymmetry factor $k = 5$.

Window size for a full pipe

The pipe is completely full when both the forward and reverse links are always busy, the forward and reverse buffers are full, and there are as many packets as possible in flight. The window size corresponding to this is denoted by W_{max}, and is given by $W_{max} = W_{full} + B_f = \mu_f T + kB_r + B_f$. When analyzing TCP evolution, we will assume that increasing the window beyond W_{max} will cause packet loss. Note that packet loss might occur at a smaller window size because the traffic is burstier than that assumed in our application of Little's law. For the running example, $W_{max} = 49$ for $k = 5$.

Forward buffer size requirement

For a fully utilized forward link, one out of k ACKs is lost, so that the average burst size into the forward buffer is k packets. Thus, $B_f \geq k$ should lead to good forward link utilization.

3.2 TCP-Tahoe evolution without random loss

The typical evolution is cyclical: the window size grows slowly during congestion avoidance to a maximum value of W_{final}, at which point there is a packet loss. This loss may or may not result in a timeout, depending on the system parameters. Subsequently, the window drops down to one, followed by rapid growth in slow start until it reaches $W_{final}/2$. At this point, the window dynamics switch to congestion avoidance until the window size reaches W_{final} again, at which point there is another loss, resulting in a new cycle. The long run throughput can therefore be computed as the number of packets successfully transmitted in a cycle, divided by the cycle duration. If a cycle

[3] In turn, for each ACK arriving back at the source, an average of λ/μ_r packets are released and arrive at the forward buffer. However, we will ignore the resulting burstiness in our calculations to obtain a simpler formula.

ends in a timeout, the length of the timeout must be included in the cycle duration, even though no packets are transmitted during most of the timeout period.

Since the slow start phase is much shorter than the congestion avoidance phase, we will ignore it for our throughput estimates. For a detailed analysis of slow start without a reverse link constraint, see [7]. The window growth in congestion avoidance is relatively slow, and is well modeled by a continuous time approximation described by a differential equation, as in [7] and [12]. In contrast to the latter references, which ignore the reverse link, here the window growth is slowed down due to the loss of ACKs on the reverse path. We will need to consider three different cases, depending on the relative size of the forward buffer and the normalized asymmetry factor:
Case 1: $k \leq B_f \leq 3k$
Case 2: $B_f < k$
Case 3: $B_f > 3k$

Case 1 can be thought of as a regime in which TCP operates normally. Case 2 corresponds to a situation in which the forward buffer is too small to handle bursts resulting from asymmetry and ACK loss, which leads to poor performance even though the forward link is very fast. Case 3 is an anomaly, in that the performance gets worse as forward buffer size becomes large. We will see that the problem that causes this can be resolved by implementing a drop from front policy at the reverse buffer. Each case is considered in detail in what follows.

Case 1: $k \leq B_f \leq 3k$

We ignore the short slow start phase in our throughput estimates. In congestion avoidance, the rate $\frac{dW}{da}$ of window growth with arriving acknowledgements is given by

$$\frac{dW}{da} = 1/W. \tag{3}$$

Assuming that the window size during congestion avoidance is large enough to keep the reverse link busy (i.e., $W \geq W_r = \mu_r T$), the rate $\frac{da}{dt}$ at which the acknowledgements arrive back at the source is given by

$$\frac{da}{dt} = \mu_r \tag{4}$$

Combining (3) and (4), we obtain that the rate $\frac{dW}{dt}$ of window growth with time is given by

$$\frac{dW}{dt} = \frac{\mu_r}{W} \tag{5}$$

which is easily solved to get

$$W(t) = \sqrt{W^2(0) + 2\mu_r t} \tag{6}$$

Denoting the window size at which packet loss occurs by $W = W_{final}$, the window size W_0 at the beginning of congestion avoidance is $W_{final}/2$. For Case 1, $W_{final} = W_{max} = \mu_f T + kB_r + B_f$. The congestion avoidance period can be divided into two phases as follows.

Phase 1: $W(t) \leq W_{full}$, so that the forward link is underutilized. The instantaneous throughput $\lambda(t)$ is given by (2) and (6). From (6), the duration of this phase is

$$t_{ca1} = \frac{W_{full}^2 - W_0^2}{2\mu_r} \tag{7}$$

($t_{ca1} = 0$ if $W_{full} \leq W_0$.)
Phase 2: $W_{full} \leq W(t) \leq W_{final}$, so that the forward link is fully utilized. The throughput $\lambda(t) = \mu_f$, and the duration of this phase is

$$t_{ca2} = \frac{W_{final}^2 - W_{full}^2}{2\mu_r} \tag{8}$$

($t_{ca2} = 0$ if $W_{final} \leq W_{full}$).

The net duration of the congestion avoidance phase is

$$t_{ca} = t_{ca1} + t_{ca2} = \frac{W_{final}^2 - W_0^2}{2\mu_r} \tag{9}$$

The average throughput is the average throughput over a cycle, and is given by

$$\bar{\lambda} = \frac{1}{t_{ca}} \int_0^{t_{ca}} \lambda(t)\ dt \tag{10}$$

Using (2), (6), and (7)-(9), we obtain upon simplification that

$$\bar{\lambda} = [\frac{2(W_{full}^3 - W_0^3)}{3(T + \frac{B_r}{\mu_r})} + \mu_f(W_{final}^2 - W_{full}^2)]/(W_{final}^2 - W_0^2) \tag{11}$$

Ideally, $\bar{\lambda}$ should be close to μ_f, the forward link capacity. As Figure 2 shows, for fixed μ_r, $\bar{\lambda}$ grows almost linearly with k (and therefore with μ_f) in the regime of Case 1.

A typical window evolution for Case 1 is shown in Figure 3. For our running example, with $k = 5 < B_f = 8$, our analysis predicts that the largest window size attained for our running example is $W_{final} = W_{max} = 49$. This matches the simulated window evolution shown in Figure 3.

Case 2: $B_f < k$

For large k, TCP performance is dominated by the ability of the forward buffer to accomodate bursts, so that the performance is relatively insensitive to k in this regime. It is therefore convenient to consider the case $k = \infty$. We begin with a qualitative explanation of TCP behavior in this regime, and then give a quantitative estimate of throughput based on modeling the deterministic evolution of TCP as random. Given the drastic nature of this approximation, the match with simulations in terms of window evolution is not perfect. However, the resulting throughput predictions are quite accurate (see Figure 2). Further study of this case is necessary to fully understand the window evolution.

Once the reverse link is fully utilized, ACKs arrive back at the source with a spacing of $1/\mu_r$. If the throughput is larger than μ_r, each ACK represents a number of successful forward packets, so that, when it is received, a burst of packets is released into the forward buffer. The maximum number of such bursts that are outstanding is simply the number of outstanding packets whose ACKs will successfully reach the source, and is equal to the maximum capacity of the reverse pipe, $W_r = \mu_r T + B_r$, which is assumed to an integer for simplicity. Since $k = \infty$, the ACKs corresponding to all packets in the burst appear at the same instant at the reverse buffer, and at most one of these ACKs can be served. This surviving ACK leads to another burst being released, with $W_r - 1$ bursts having been released between this burst and the earlier burst being ACKed.

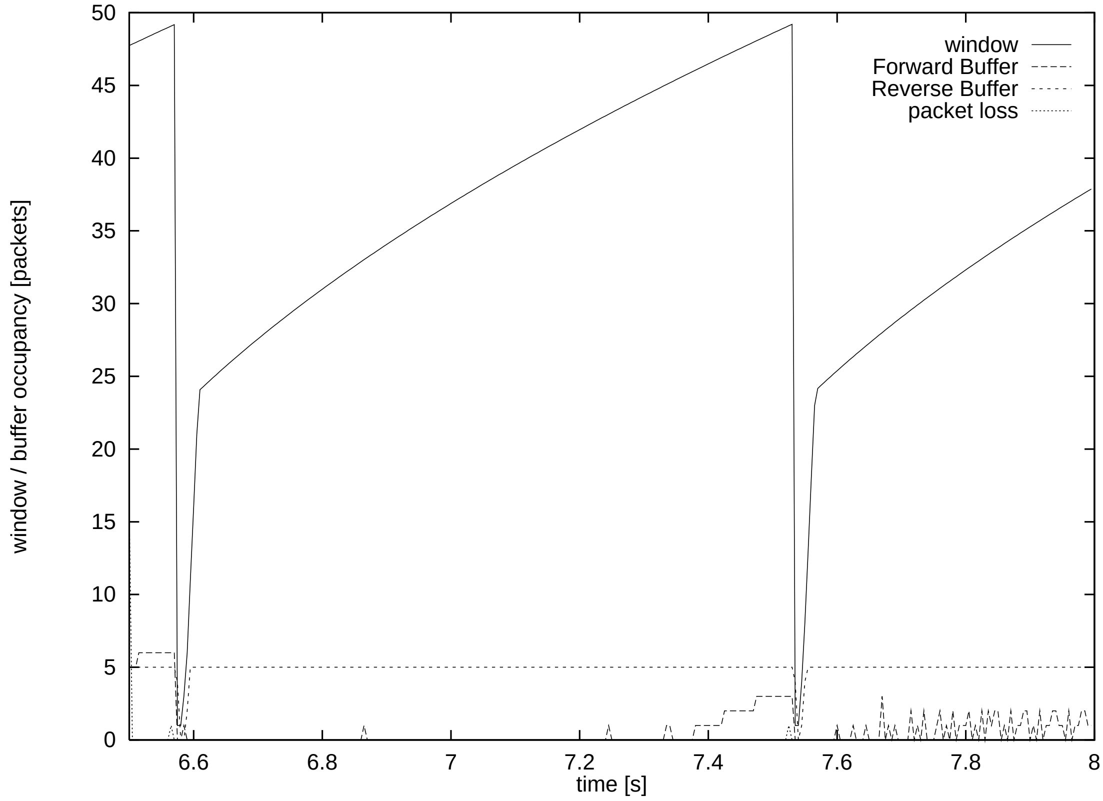

Figure 3: Simulation of the window evolution for the running example with TCP-Tahoe when $k = 5$. Similar evolutions are seen for all k satisfying the condition in Case 1.

These two bursts (and every W_rth burst before and after them) can therefore be thought of as the same burst circulating in the path of the connection. With such an identification, there are W_r different bursts, and we denote by X_i the current size of the ith burst, $i = 1, ..., W_r$. The current window size is therefore $W = \sum_{i=1}^{W_r} X_i$. Suppose now that the window size gets incremented from W to $W+1$ when an ACK from burst i arrives. Then X_i is incremented to $X_i + 1$ (i.e., one more packet is added to the burst). In congestion avoidance, the next increment occurs after $W+1$ more ACKs have been received by the source, i.e., once the ACKs for $W+1$ further bursts have arrived. Thus, in terms of our modulo W_r indexing scheme, the next burst that gets incremented has index

$$j = (i + W + 1) \text{ modulo } W_r \tag{12}$$

The preceding can be used to deduce the deterministic evolution of the window and burst sizes. Clearly, some bursts could be incremented more often than others during this evolution. When the burst with the largest size has more than $B_f + 1$ packets, there is a buffer overflow and the cycle ends. We would now like to predict the window size W_{final} at which this happens, after which the analysis of Case 1 applies for estimating the throughput. The successive choices of indices dictated by (12) appears chaotic, but more study of this map (which in itself is an approximation to actual TCP behavior) is necessary. For our purpose here, we approximate the evolution due to (12) by the following random evolution: for each increment of the window size, the burst whose size is to be increased by one is chosen uniformly at random among the W_r bursts. Let $X_k^{(n)}$ denote the size of the kth burst after the nth window increment, where $k = 1, ..., W_r$, and where we assume that $X_k^{(0)} \equiv 1$ when the reverse link first gets fully utilized (i.e., when $W = W_r$). Let I_n denote the index of the burst that is incremented for the nth window increment. We will assume that the I_n are independent and identically distributed random variables which are uniformly distributed on $\{1, ..., W_r\}$. Then the evolution of the burst sizes is given by

$$X_k^{(n)} = \begin{cases} X_k^{(n-1)} + 1 & k = I_n \\ X_k^{(n-1)} & else \end{cases}$$

The size of the largest burst after the nth increment is denoted by $X_{max}^{(n)} = max_k X_k^{(n)}$. In order to estimate the window size reached before the largest burst exceeds $B_f + 1$, we consider the following criterion: Let n^* denote the smallest n such that

$$P[X_{max}^{(n)} > B_f + 1] \geq 0.5 \tag{13}$$

This is the median number of increments before a buffer overflow due to the maximum sized burst. The window size corresponding to this is $W^* = n^* + W_r$, and this is taken to be W_{final} in our throughput estimates.

We estimate n^* by using a union bound for the left-hand side of (13):

$$P[X_{max}^{(n)} > B_f + 1] \leq \bigcup_{k=1}^{W_r} P[X_k^{(n)} > B_f + 1] = W_r P[X_1^{(n)} > B_f + 1] \tag{14}$$

using the fact that the $\{X_k^{(n)}\}$ are identically distributed (though not independent). Clearly, $X_1^{(n)} - X_1^{(0)} = X_1^{(n)} - 1$ is binomially distributed with parameters n and $1/W_r$, so that the extreme right-hand side of (14) can be computed by summing over the tail of the distribution. Actually, the first term in the tail is a good approximation to the entire tail, and the computation is further simplified by using the Poisson approximation to the binomial distribution. We finally obtain that

$$P[X_{max}^{(n)} > B_f + 1] \approx W_r e^{-n/W_r} \frac{(n/W_r)^{B_f+1}}{(B_f + 1)!} \tag{15}$$

For $k > B_f$, the W_{final} predicted by the preceding computations for our running example is 54.5. As shown in the Figure 4, the actual maximum window size reached in the cycle is 45 for $k = 10$ (recall that $B_f = 8$).

It is natural to ask why burstiness does not determine the performance for Case 1 ($B_f \geq k$), for which $W_{final} = W_{max}$, corresponding to a full pipe. In this case, the buffer can accomodate bursts of size upto $k + 1$ (since the buffer capacity is $k + 1$, including the packet in service). When $X_{max}^{(n)}$ first goes from k to $k + 1$, at least two ACKs from this burst are served (since at least one ACK packet is transmitted for every k packets transmitted on the forward link). This results in the large burst getting broken up into smaller bursts. This is probably what enables all bursts to maintain roughly equal sizes, so that buffer overflow occurs due to a full pipe rather than due to large burst sizes.

Case 3: $B_f > 3k$

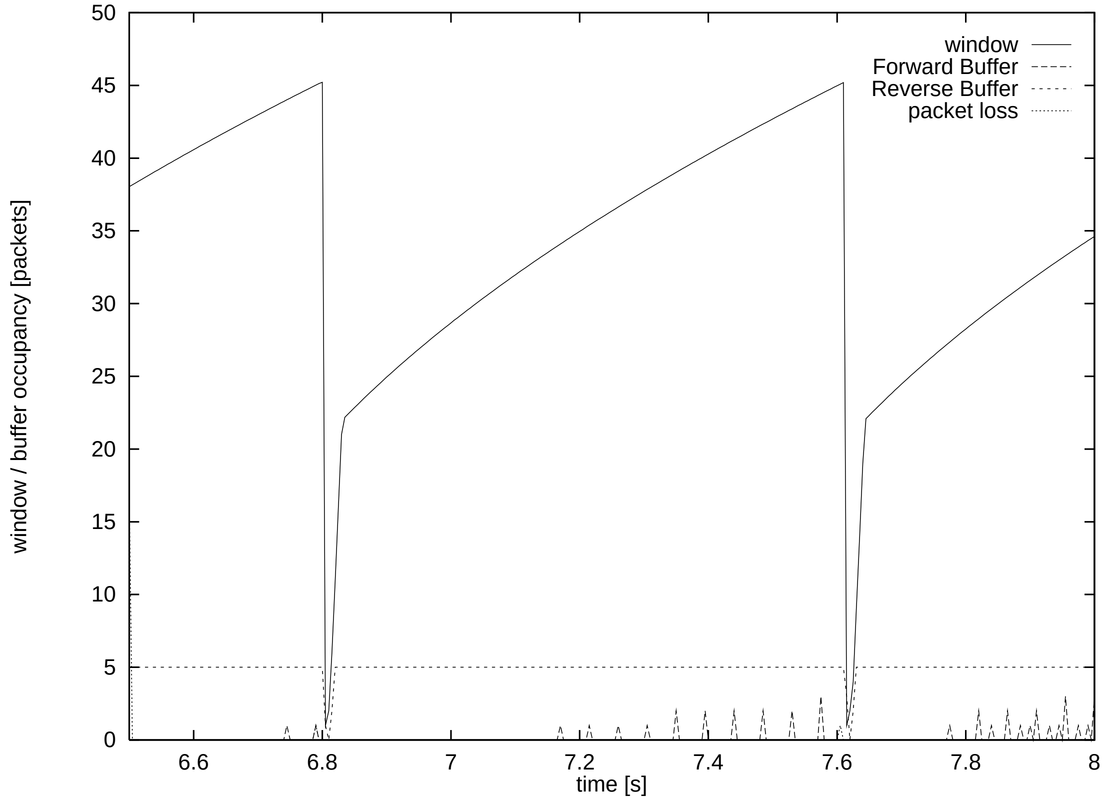

Figure 4: Simulation of the window evolution for the running example with TCP-Tahoe when $k > B_f = 8$. The evolution shown is for $k = 10$, but similar evolutions are seen for all $k > B_f$.

The evolution here is exactly as in Case 1, except that a timeout occurs at the end of congestion avoidance because of a failure of the fast retransmit option. Consider a packet loss that occurs when

$$W_{final} = W_{max} = \mu_f T + kB_r + B_f$$

If packet n is lost, once all packets upto $n-1$ are ACKed, the outstanding packets are $n, n+1, ..., n + W_{final} - 1$. For all packets among these that are successful, the destination generates cumulative ACKs saying "next expected = n." After 3 duplicate ACKs arrive at the source, packet n is retransmitted and the window is dropped to one, so that no new transmissions are attempted. However, by the time 3 duplicate ACKs are received (spaced by $1/\mu_r$), the forward buffer is reduced by $3k$. If this does not drain the forward buffer (which will happen if $B_f > 3k$), the reverse link will remain fully utilized, and the reverse buffer will be full by the time the ACK for the retransmission arrives at the reverse buffer, so that the latter gets lost. When this happens, the source keeps waiting for the ACK to arrive until there is a timeout. This can be thought of as Phase 3, and is of duration t_o, the length of the timeout period. Prior to this, Phase 1 and Phase 2 of the evolution are exactly as in Case 1.

With the durations t_{ca1} and t_{ca2} as in (7)-(8), the net duration of congestion avoidance becomes

$$t_{ca} = t_{ca1} + t_{ca2} + t_o$$

and the average throughput is obtained by plugging into the formula (10), noting that $\lambda(t) = 0$ in Phase 3, since the window size is one and there is an unacknowledged packet.

A simple change that prevents the timeout from happening is to implement a drop from front policy at the receive buffer, which makes sure that the latest ACK information gets through to the source. This works because, with cumulative ACKs, the amount of information carried by later ACKs is always more than that contained by earlier ACKs. Simulations show that implementation of this policy eliminates all timeouts in Case 3, so that the throughput formula becomes similar to that in Case 1. Note that drop from front has been previously proposed for the forward path for TCP over ATM for an entirely different reason [9].

Typical window evolutions without and with drop from front are shown in Figures 5 and 6, respectively. The time over which the window is constant at one in Figure 5 corresponds to the timeout period.

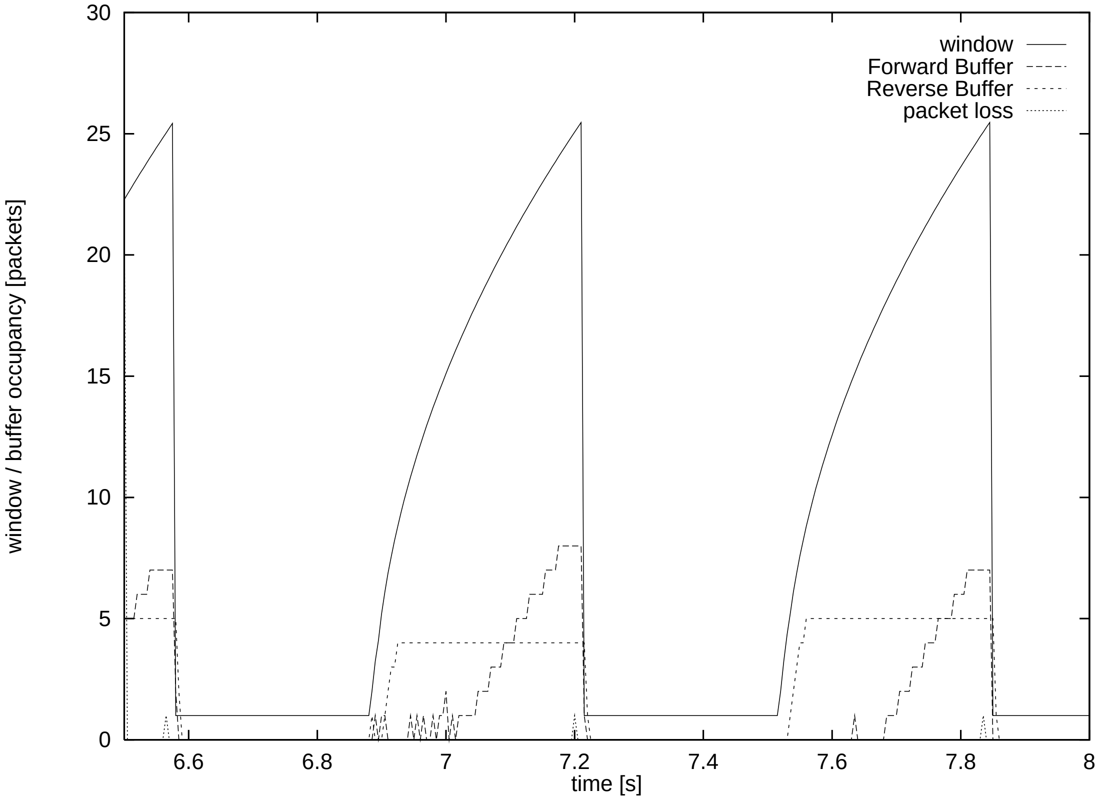

Figure 5: Simulation of the window evolution for the running example with TCP-Tahoe when $k = 2$ with a FIFO reverse buffer. The window size is constant at one for the duration of the timeout at the end of the cycle.

3.3 TCP-Reno evolution without random loss

Cases 1 and 3: $B_f \geq k$

The evolution of TCP-Reno is similar to that for TCP-Tahoe, except that recovering from a packet loss is more problematic for TCP-Reno because of the slow reverse link. Suppose packet n

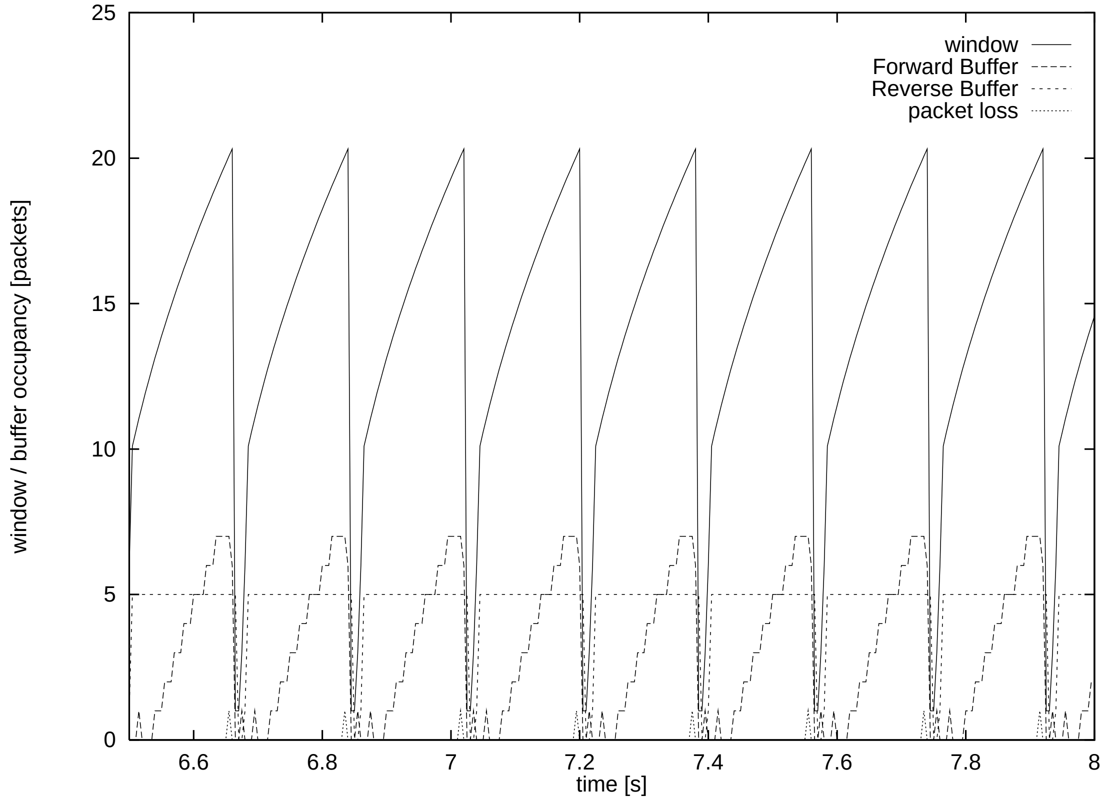

Figure 6: Simulation of the window evolution for the running example with TCP-Tahoe when $k = 2$ and drop from front is implemented at the reverse buffer. Note that the timeout is eliminated.

is lost at $W = W_{final}$, and suppose that all packets sent before and after packet n get through. Thus, ACKs for packets $n+1, ..., n+W_{final}-1$ will be generated by the destination, each saying "next expected $= n$." After three such duplicate ACKs are received by the source, packet n is retransmitted, and the window is cut back to $W_{final}/2$. Further, in order to avoid a burst when the ACK for the retransmitted packet gets through, after $W_{final}/2$ duplicate ACKs arrive, TCP-Reno starts transmitting a packet for each further duplicate ACK that arrives even though that is not allowed according to the window-based protocol. If the reverse link were not a bottleneck, this would enable transmission of roughly $W_{final}/2$ new packets (which is equal to the new window size) by the time the ACK for the retransmission gets back, which means that there would be no burst of packets at that point. However, due to the slow reverse link, by the time the retransmitted ACK gets back, the number of duplicate ACKs received is roughly only $\mu_r T + 3$ (since ACKs can arrive back at a rate no faster than μ_r), even though actually $W_{final} - 1$ packets have been served. Since this is typically smaller than $W_{final}/2$, no new packets are released, so that when the ACK for the retransmitted packet is finally received, a burst of $W_{final}/2$ packets is released into the forward buffer by the source. This is larger than the forward buffer size B_f for most situations of interest, so that multiple packet losses occur. This causes a window cutback to $W_{final}/4$. Further, fast retransmit using cumulative ACKs is only able to recover from very small bursts of multiple packet loss, so that a timeout eventually occurs.

Since the timeout at the end of congestion avoidance occurs at $W = W_{final}/4$, the slow start phase in the next cycle ends at $W_{final}/8$. Thus, congestion avoidance starts at $W_0 = W_{final}/8$ and continues till W_{final}, being terminated by a timeout. The timeout occurs for both Case 1 and Case 3 (i.e., for all $B_f \geq k$), and the average throughput is computed as in Case 3 for TCP-Tahoe. Since congestion avoidance begins at a smaller window value for TCP-Reno, which causes it to have a smaller throughput than TCP-Tahoe. An illustrative window evolution is shown for $k = 5$. As for TCP-Tahoe, the analysis predicts $W_{final} = 49$. Note that there are two consecutive window cutbacks, as well as a timeout, at the end of the cycle.

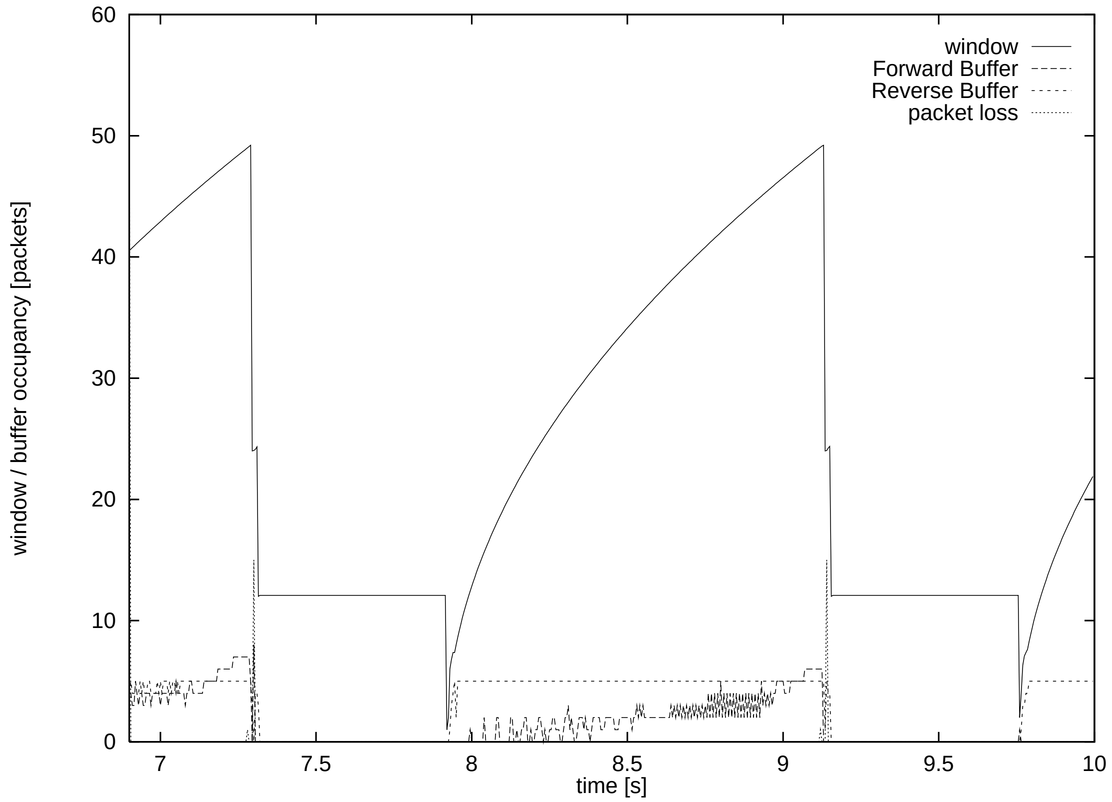

Figure 7: Simulation of the window evolution for the running example with TCP-Reno when $k = 5$. Note that there is a timeout at the end of the cycle, unlike Case 1 for TCP-Tahoe.

Case 2: $B_f < k$

The behavior here is similar to Case 2 for TCP-Tahoe, except for the following caveat. If W_{final} is the window size when loss occurs due to a burst, slow start ends at $W = W_{final}/8$. Since the bursts are formed at the slow start phase, if $W_{final}/8 < W_r$, then there are only $W_{final}/8$ bursts circulating and getting incremented (pseudo)randomly during congestion avoidance. Let n_b denote the number of bursts. Then, if the cycle terminates after n increments during congestion avoidance, then $W_{final} = n + n_b$, and $n_b = min(W_{final}/8, W_r$, which gives $n_b = min(n/7, W_r)$. Replacing W_r by n_b in Case 2 of TCP-Tahoe as before, we obtain n^* and thus W_{final}. We then apply Case 1 of TCP-Reno to compute the throughput. For our running example, the analysis predicts $W_{final} = 39.4$, which, while different from the window evolution simulated in Figure 8, is good enough to yield a reasonable throughput prediction (see Figure 2).

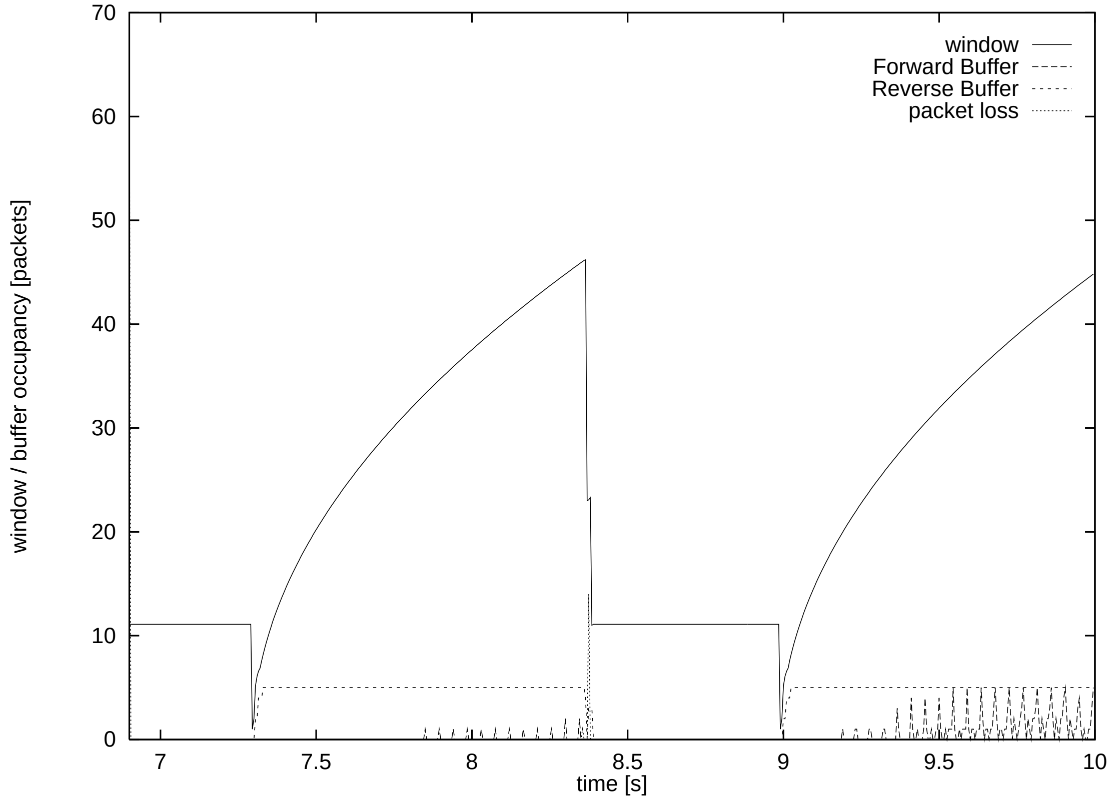

Figure 8: Simulation of the window evolution for the running example with TCP-Reno when $k > B_f = 8$. The evolution shown is for $k = 10$, but similar evolutions are seen for all $k > B_f$.

3.4 Evolution with random loss

According to our loss model, any packet served at the forward link may be lost with probability q, and such losses are independent. To focus on the effect of random loss, we consider the normal operating condition for TCP-Tahoe, which corresponds to Case 1 ($k \leq B_f \leq 3k$) in Section 3.2. The effect of random loss on TCP performance has been considered previously in [7], [8], for a model in which the reverse path is ignored (i.e., assumed to be fast and lossless). The result derived in [7], [8], was that TCP throughput deteriorates significantly due to random loss if $q(\mu_f T)^2$ is large (say 10 or more). An alternative interpretation is that a throughput λ_f can be attained if $q(\lambda_f T)^2$ is of the order of 1 or less. That is, the attained throughput λ_f is proportional to $\frac{1}{T\sqrt{q}}\}$. Here, we use similar arguments to show that the performance is further affected by the slow reverse link: the throughput deteriorates if $kq(\mu_f T)^2$ is large (or, in other words, the attained throughput is proportional to $\frac{1}{T\sqrt{kq}}\}$). Thus, with identical forward path loss, the throughput degradation is significantly worse for TCP flows with slow reverse paths than for TCP flows which do not have a reverse path bottleneck (or for TCP-friendly flows with rate reduction determined by the loss-throughput relation obtained by considering only forward path bottlenecks).

The intuition behind the preceding threshold rule is as follows. If the loss probability is low enough, the likelihood of a random loss rather than a buffer overflow terminating a cycle is small, so that the evolution is similar to that without loss. On the other hand, if the loss probability is high, packet losses occur relatively early in a cycle, and result in small initial values for the congestion avoidance phase that follows. This results in small window sizes (determined by random losses rather than congestion) throughout the congestion avoidance phase, and therefore causes low link utilization. Since window growth is timed by ACKs arriving at the source, a slow reverse link results in slower window growth after a loss, causing even worse utilization.

Our objective is to determine when random loss causes the congestion avoidance phase to have a significantly smaller average window size than that for a lossless evolution. We ignore the slow start phase for the purpose of arriving at the threshold rule: since the slow start phase is short, the probability of a random loss occurring within this phase is small unless the loss probability is already very high. Note that the evolution of a TCP-Tahoe cycle (defined now as the window evolution between successive window cutbacks), is no longer periodic: the window size at which a loss occurs for a given cycle, and therefore the window size at which congestion avoidance begins in the next cycle, is random. Since characterizing the Markovian evolution that results is complicated, and offers little insight, we resort to a "fixed point" approximation in an attempt to characterize the "average" behavior of a TCP cycle due to random loss. Let w_q be the average window size at the beginning of congestion avoidance. In order for the congestion avoidance phase in the next cycle to also begin with w_q, the present congestion avoidance cycle should end with a loss at a window size of $2w_q$. We would like to choose w_q such that exactly $1/q$ packets (which is the average number of packets transmitted between losses for a loss probability of q) are transmitted on the forward path by the time the window evolves from w_q to $2w_q$. If the resulting value of $2w_q$ is found to be much smaller than the window size $W_{max} = \mu_f T + B_f + kB_r$ for lossless evolution, then random loss is expected to significantly reduce throughput.

To find w_q, we use the differential equations in Section 3.2 to model TCP evolution in congestion avoidance as before. Assuming that the reverse link is fully utilized throughout congestion avoidance, we obtain as before that

$$W(t) = \sqrt{W^2(0) + 2\mu_r t}$$

where $W(0) = w_q$ for our fixed point approximation. If t_q is the duration of the congestion avoidance phase, we have $W(t_q) = 2w_q$. The number of packets transmitted during this time is estimated by

$$n_q = \int_0^{t_q} \frac{\sqrt{w_q^2 + 2\mu_r t}}{T + \frac{B_r}{\mu_r}}\, dt = \frac{7w_q^3}{3(\mu_r T + B_r)}$$

Setting $n_q = \frac{1}{q}$, we obtain that

$$w_q^3 = \frac{3(\mu_r T + B_r)}{7q} \tag{16}$$

In order for random loss not to affect performance significantly, we should have $2w_q$ equal or exceed the maximum window size for lossless evolution, $W_{max} = \mu_f T + B_f + kB_r$. This results in the following criterion:

$$q\frac{(\mu_f T + B_f + kB_r)^3}{\mu_r T + B_r} \leq 24/7$$

The left-hand side is lower bounded by $kq(\mu_f T + B_f + kB_r)^2$. If B_f and kB_r scale with the forward bandwidth-delay product $\mu_f T$, we arrive at a threshold rule in $kq(\mu_f T)^2$ as promised. When this quantity is large, throughput is expected to deteriorate as compared to lossless performance.

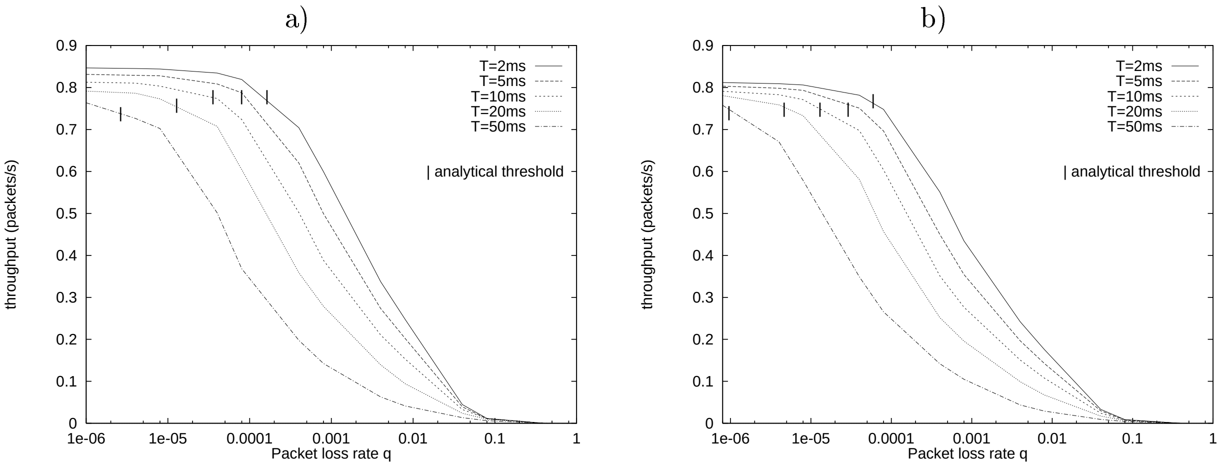

Figure 9: Throughput as a function of loss probability q for the modified running example using $B_f = 10$ when $k = 5$ (a) and $k = 7$ (b). The mark on the curves represents $kq(\mu_f T)^2 = 1$

Figure 9 shows the throughput as a function of random loss for different values of k and τ (note that the round-trip time T is larger than τ). Note the drastic decrease in throughput when the loss probability is an order of magnitude higher than the analytically derived threshold values marked on the curves.

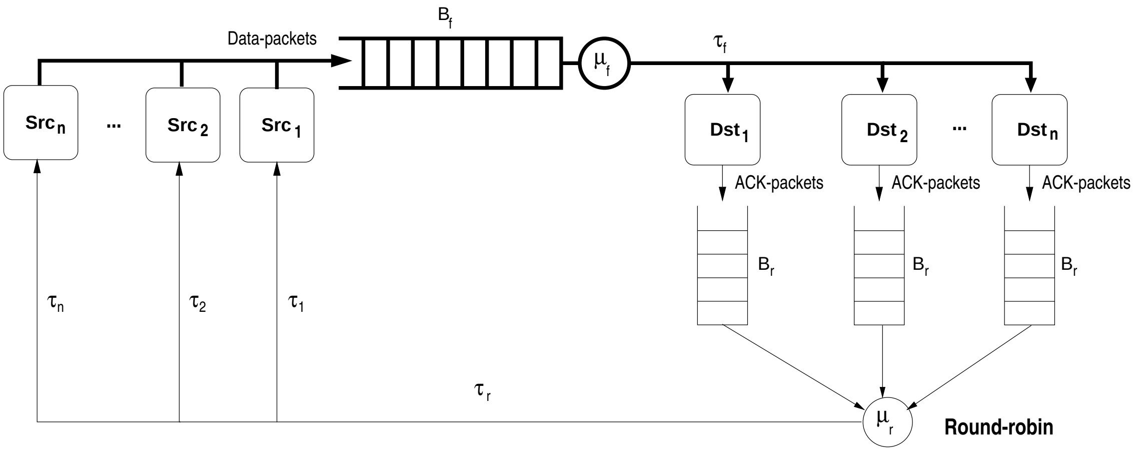

Figure 10: Asymmetric system model with per connection queues at the reverse link

4 Multiple TCP Connections

We consider multiple TCP connections which share the forward link and buffer using a FIFO service discipline. Even without a slow reverse link, FIFO sharing of the forward link lead to unfairness among forward connections with different round trip times [8]. We will find that these effects are exacerbated by asymmetry. For bidirectional traffic (both forward and reverse connections), the situation is even worse. In this case, the slow reverse link serves both as a data path (for reverse connections) and an ACK path (for forward connections). We find that, if the reverse link buffer is FIFO, then reverse connections are almost completely shut out. This problem can be alleviated by using per connection weighted round robin queueing (or fair queueing) at the reverse link, but the weights must be carefully chosen so as not to compromise the performance of the forward connections. Intermediate options for sharing the reverse link include using several FIFO buffers, each shared by several connections, served in round robin fashion. Such a scheme could work quite well if forward and reverse connections are served in different buffers. In our simulations, we consider only the two extreme service options for the reverse link: FIFO and per connection weighted round robin service.

4.1 FIFO Reverse Link Service

FIFO service would be the natural option when several hosts are connected to the same network interface (such as a cable modem). We have seen in Section 3 that ACK losses are very high (as high as $k-1$ out of k) when $k > 1$. This leads to adverse interactions when multiple TCP connections fully share the reverse buffer.

4.1.1 Forward and Reverse Connections

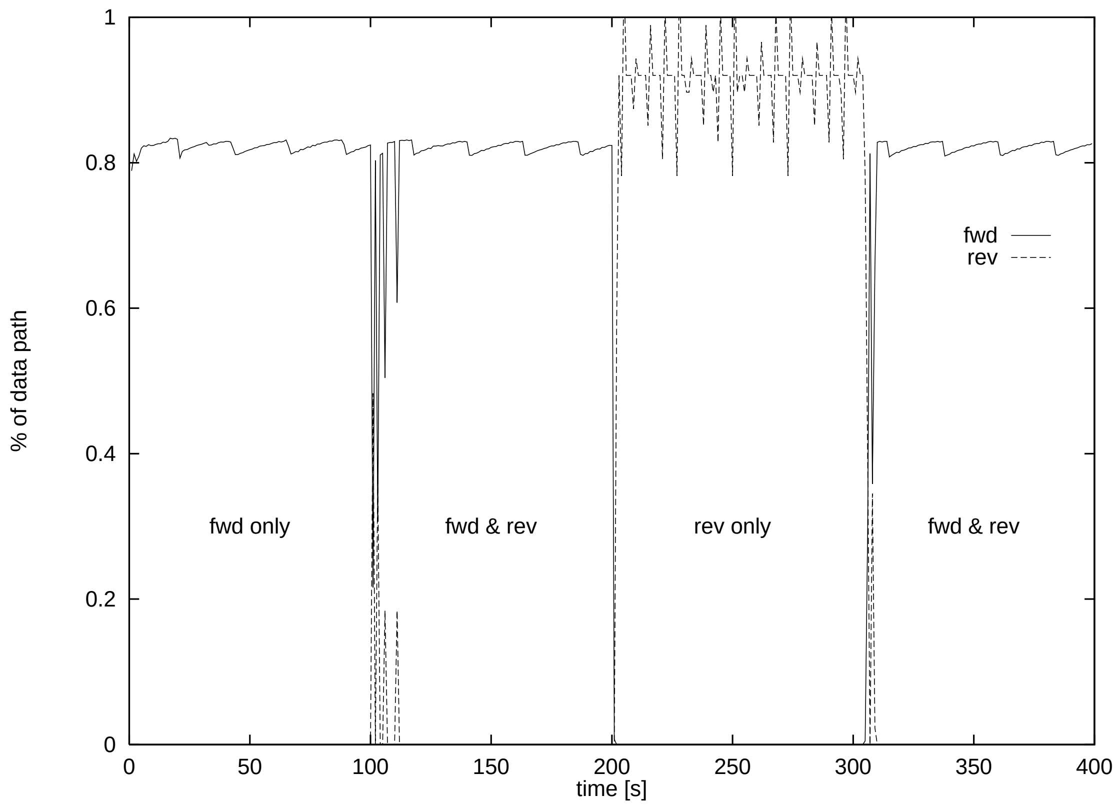

Figure 11: A forward and a reverse connection with a fully shared upstream buffer. ($\mu_r = 1000$ pkt/s, $k = 5$, $\tau_f = \tau_r = 1ms$, $B_f = 10$, $B_r = 5$)

Figure 11 shows throughputs (as percentages of their maximum data path bandwidths) obtained by one forward and one reverse TCP connection. The buffer sizes indicated in the figure give the total buffering available for ACK and data packets (one buffer location is consumed irrespective of whether the packet is an ACK or a data packet).

Initially, the forward connection is the only connection. It achieves a throughput of 84% and the upstream link buffer is full with ACKs being sent by this connection. When the reverse connection starts, it experiences a high packet loss in the upstream link buffer. This forces this connection to operate using small windows and high timer backoff values, leading to poor throughput. The forward connection cuts down its rate only if there is packet loss in the forward path, or if it loses a whole window's worth of ACKs in the reverse path. Since neither of these events happens often enough, the reverse connection gets high throughput only after the forward connection ends at $t = 200$s. Thus, FIFO reverse link service leads to poor performance for the reverse connection, so that some means of guaranteeing a portion of the link capacity to the latter appears to be necessary. One possibility is to "thin" the ACKs for the forward connections *before* they reach the reverse link: this could be done at the TCP layer by the destination, or at the network layer by a *TCP-aware* agent. The drawback of ACK thinning is that it violates protocol layering: either the transport layer must be aware of (possibly time-varying) asymmetry at the network layer, or the network layer must be specifically engineered for the TCP application. Another possibility, which has the advantage of requiring less interaction between protocol layers at the expense of additional complexity, is to use per connection queueing at the reverse link. This option is explored in Section 4.2.

4.1.2 Forward Connections Only

Figure 12 shows how four TCP-Tahoe connections with equal round-trip times share bandwidth for a system with $k = 5$. The plot shows the fraction of the forward connection utilized by the different connections as a function of connection time. Over short intervals (tens of seconds), increasing cumulative throughput for a given connection indicates larger instantaneous throughput, while decreasing cumulative throughput indicates smaller instantaneous throughput. It is seen from the figure that, while the cumulative throughputs for all connections converge slowly to equal values, the link sharing is grossly unfair (i.e., there are large variations in the instantaneous throughputs) over time scales of the order of tens of seconds. This short term unfairness is in contrast with earlier results on TCP performance without ACK path congestion [8], for which connections with equal round trip times typically see a synchronized evolution which is roughly fair in both the short and long term.

An explanation for the short-term unfairness is as follows. Assume connections start at slightly different times. The first active connection saturates the reverse path and because of the high $1 - 1/k$ ACK loss rate, a newly started connection will almost always lose the ACK corresponding to its first packet. This causes a time-out and a backoff of the timer. The retransmit faces the same situation resulting in very large timer values and large periods of idleness (of the order of tens of seconds because of the large timer backoffs) for all connections other than the one currently saturating the reverse link. This continues till a loss in the forward path for the currently active connection reduces that connection's window to one (with a timeout occurring in some cases, as seen in Section 3). One of the idle connections now becomes the dominant connection. Over the very long term connections tend to share the forward link fairly but over intervals of tens of seconds

(500ms timer granularity multiplied by backoff values of 32 or 64), one of the connections gets most of the bandwidth. If connections are used for long file transfers, a connection will see virtually no progress for tens and possibly hundreds of seconds (depending on the number of competing connections) and then get a period of high bandwidth (corresponding to a few TCP cycles for the single connnection case analyzed in in Section 3). If the file transfer does not complete in one of these periods, another pause of 10 to 100s of seconds will follow.

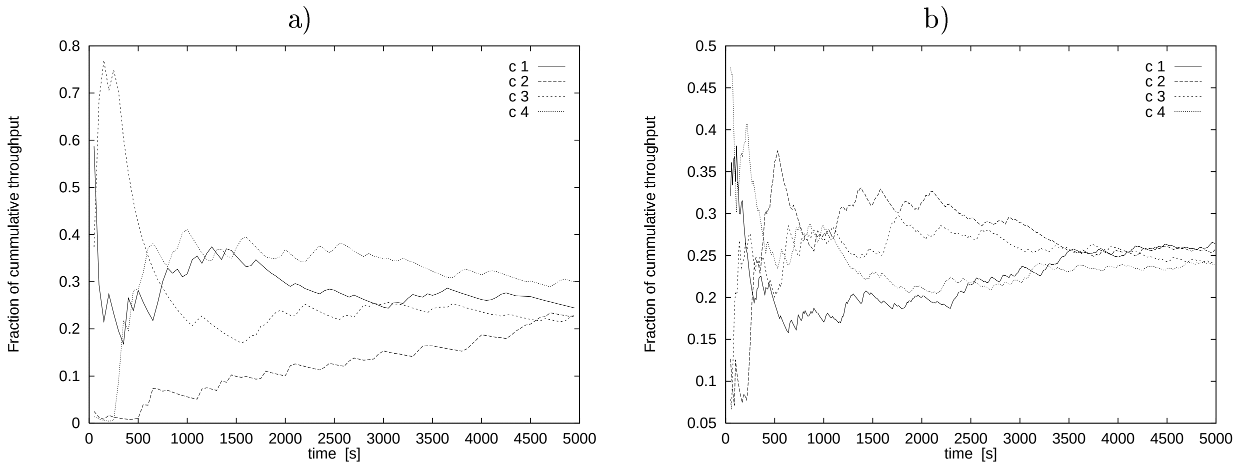

Figure 12: Temporary lockouts between 4 connections with equal propagation delays for a fully shared upstream buffer: (a) TCP-Tahoe (b) TCP-Reno. ($\mu_r = 1000$ pkt/s, $k = 5$, $\tau_f = \tau_r = 1ms$, $B_f = 10$, $B_r = 5$)

Unequal round trip times exacerbates the problem of unfairness. Simulations show that some connections get shut out for long periods. Also, the system evolution is highly sensitive to slight changes in packet arrival times, burst lengths and system parameters. Figure 13 illustrates this for a system carrying 4 forward connections with different round trip times. Connection 1 with 4 ms round trip time gets most of the link bandwidth for a very long time while other connections are almost shut out. This apparently stable situation changes suddenly with connection 4 now getting increasing bandwidth share. Connections 2 and 3 still remain almost shut out.

We conclude that FIFO sharing of the reverse link leads to unfair and unpredictable behavior, and is not suitable for supporting TCP over an asymmetric network.

4.2 Round Robin Reverse Link Service

Round robin service should not be cumbersome to implement in systems where connections access the network using different network access devices, but share the same reverse link (e.g. an HFC upstream channel). We assume throughout that a drop from front queueing discipline is adopted for acks on the reverse link (as shown in [9], a drop from front strategy is beneficial even for data packets), in order to feed back the most recent ack information to the TCP source. Such a strategy, coupled with the ack loss on the slow reverse link, automatically results in the appropriate amount of ack thinning, so that explicit ack thinning at the TCP destination is not required.

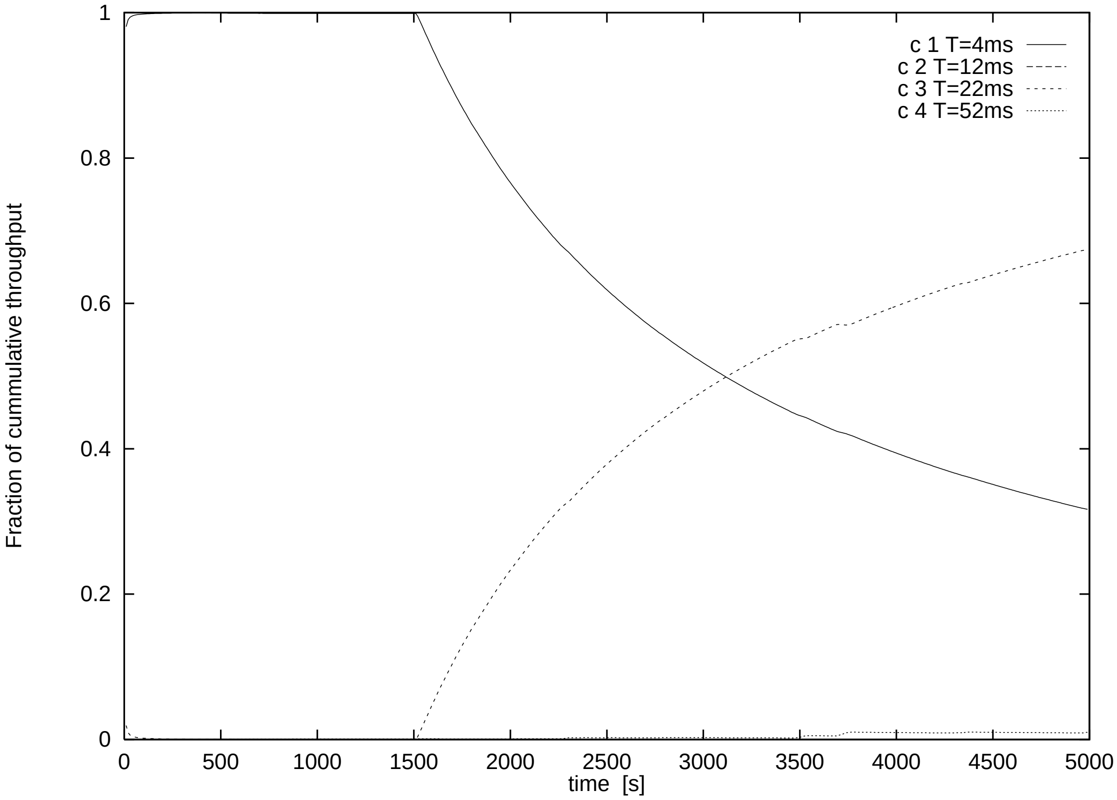

Figure 13: 4 forward connections with different propagation delays and a fully shared upstream buffer. ($\mu_r = 1000$ pkt/s, $k = 5$, $B_f = 10$, $B_r = 5$)

4.2.1 Forward and Reverse Connections

The purpose of weighted round robin service on the reverse link is to prevent the lockout of reverse connections due to the high volume of acks generated by the forward connections. However, providing a bandwidth guarantee for reverse connections leads to two new issues that must be addressed in order to guarantee good performance for forward connections.

Issue 1: Effective Increase in Normalized Asymmetry: If a forward connection is allocated a fraction α of the reverse link bandwidth (where $\alpha < 1$), its *effective* normalized asymmetry increases by a factor of $\frac{1}{\alpha}$. Thus, for full utilization of the forward link, the analysis in Section 3 implies that we need a larger forward buffer size, satisfying $B_f \geq \frac{k}{\alpha}$, or equivalently, that we must allocate at least a fraction $\alpha = \frac{k}{B_f}$ (assuming $B_f \geq k$) of the reverse link bandwidth to the forward connection.

Issue 2: Ack Starvation on the Reverse Link: The reasoning in the preceding paragraph assumes that the forward connection sees a steady service of $\alpha\mu_r$ acks per second on the reverse link. In practice, data packets for the reverse connections can be much larger in size than ack packets for forward connections, which may result in bursty service for the acks. Consider, for example, the case of one forward and one reverse connection. Suppose that the data packets of the forward connection are a_f times larger than the ack packets for the forward connection, and that the data packets for the reverse connection are a_r times larger than the ack packets for the forward connection. If a complete data packet for the reverse connection is served without preemption on the reverse link, there is an *ack starvation period* of duration $\frac{a_r}{\mu_r}$ during which no acks reach the TCP source for the forward connection. In order to keep the forward link busy through this starvation period, there should be $\mu_f \frac{a_r}{\mu_r}$ data packets for the forward connection queued up in the forward buffer at the beginning of the starvation period. But this can only be possible if $B_f \geq \mu_f \frac{a_r}{\mu_r}$. Thus, if $a_f = a_r \gg 1$, then the forward buffer must be larger than the *raw* asymmetry (rather than the normalized asymmetry) for full link utilization, which is too stringent a requirement for most practical applications. The only solution to this problem is to make a_r small. This can be done by breaking up a large TCP data packets for the reverse connection into smaller segments (ideally, of a size equal to the ack packets for the forward connection, so that $a_r = 1$) in some manner prior to allowing it access to the round robin server. Such segmentation could either be done explicitly at the network or data link layer, or it could be emulated by allowing some form of preemptive transmission of ack packets (e.g., interrupt transmission of a long data packet if there is an ack packet to be served).

Figure 14 shows the performance impact of the ack starvation phenomenon mentioned above. One forward and one reverse connection is considered, and each connection is allocated half the bandwidth on the reverse link. Thus, the effective asymmetry for the forward connection increases by a factor of two. Thus, if we double the size of the forward buffer, we expect to see the same throughput characteristic as in Figure 2, where there was no reverse connection (i.e., we should now expect the throughput to drop to the $k = \infty$ plateau at $k \approx B_f/2 = 7.5$). As shown in Figure 14, this is indeed the case when when the data packets for the reverse connection are segmented into chunks of 20 bytes (smaller than the size of the ack packets for the forward connection) before being allowed access to the round robin server.[4] However, when we allow data packets of size 1000 bytes to be served nonpremptively, the ack starvation seen by the forward connection causes the throughput to plateau at 1000 packets/second as we increase the forward link speed. We are unable to offer any explanation for this particular value of the plateau, and leave performance analysis in the presence of ack starvation as an open problem. However, comparing the two plots in Figure 14, it is clear that ack starvation causes a severe performance penalty.

4.2.2 Forward Connections Only

Figure 15 shows that, for four forward connections with equal round trip times, per connection reverse link queueing and round robin service eliminates the lockout phenomenon with FIFO reverse link queueing seen in Figure 12, and restores throughput fairness both in the short and long terms. For connections with unequal round trip times, it can be seen from Figure 16 that per connection queueing eliminates the lockout of some connections shown for FIFO queueing shown in Figure 13. Of course, the inherent unfairness of TCP towards connections with longer round trip times still persists. This could be remedied by using per connection queueing and round robin service on on the forward path [8] as well.

[4]In fact, we get somewhat better performance than predicted, because the reverse connection does not always fully utilize its share of the reverse link bandwidth, so that the effective normalized asymmetry does not quite double.

5 Concluding Remarks

Our study of the consequences of a slow reverse link yields the following conclusions and recommendations:
(1) For a single forward connection, the performance of TCP-Tahoe is superior to that of TCP-Reno. This is because the slow reverse link prevents TCP-Reno's loss recovery mechanism from functioning as designed, resulting in a coarse-grained timeout after every loss.
(2) In order to obtain good utilization of the forward link, it is essential that the forward link buffer size B_f is at least equal to the normalized asymmetry k. Indeed, if B_f is fixed and the forward link speed is increased, the throughput is ultimately limited by B_f, so that the link utilization goes to zero.
(3) For large forward buffers ($B_f > 3k$), TCP-Tahoe may also incur a timeout after every cycle, resulting in throughput loss (in general, replace 3 by the threshold for duplicate ACKs used in the TCP fast retransmit mechanism). This problem can be eliminated by implementing drop from front at the reverse buffer. Doing this restores a monotonic improvement of throughput with forward buffer size.
(4) Asymmetry increases TCP's already high sensitivity to random loss anywhere in the forward path of the TCP connection. Specifically, we show that random loss leads to significant throughput deterioration when the product of the loss probability, the normalized asymmetry, and the square of the bandwidth-delay product exceeds a threshold.
(5) For multiple connections using FIFO queueing on the reverse link, poor performance results both for undirectional (forward connections only) and bidirectional (forward and reverse connections) traffic. For multiple forward connections, small changes in timing can cause a connection to be locked out for tens or even hundreds of seconds. Reverse connections whose data packets share the slow reverse link with ACKs from forward TCP connections can get shut out, since data packet loss due to congestion on the reverse link is much more damaging than loss of cumulative ACKs. Round robin service on the reverse link procides a solution to these problems.
6) We recommend weighted round robin service with drop from front queueing for each connection (or connection class) on the reverse link, for both unidirectional and bidirectional traffic. There is no need for explicit ack thinning by the TCP destination, since acks are thinned implicitly by the slow reverse link, and the latest ack information is preserved using drop from front queueing. For unidirectional traffic (forward connections only), round robin service eliminates the extended lockout periods seen with FIFO queueing. For bidirectional traffic, round robin service provides good performance for both forward and reverse connections, provided that the weights are chosen appropriately, and provided that the data packets for the reverse connections are segmented into smaller chunks before being allowed access to the round robin server.
7) Serving large data packets from reverse connections nonpremptively leads to the phenomenon of ack starvation, and severely limits the utilization of the forward link by forward connections. We strongly recommend either implicit or explicit segmentation of data packets (for reverse connections) on the reverse link, ideally to a size equal to that of the ack packets (for forward connections).

Problems that merit further investigation include analysis of the behavior of multiple connections and design of MAC protocols and study of their effect on TCP performance.

References

[1] 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.

[2] 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.)

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

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

[5] 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.

[6] A. Kumar *Preprint 1997*

[7] T. V. Lakshman and U. Madhow, "Performance Analysis of Window-Based Flow Control using TCP/IP: the Effect of High Bandwidth-Delay Products and Random Loss" *IFIP Transactions C-26, High Performance Networking V*, pp. 135-150, North-Holland, 1994.

[8] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Networking,* vol. 5, no. 3, pp. 336-350, June 1997.

[9] T. V. Lakshman, A. Neidhardt and T. J. Ott, "The Drop from Front Strategy in TCP over ATM and its Interworking with other Control Features" *Proc. Infocom '96*, pp. 1242-1250

[10] T. J. Ott, M. Mathis and J. H. B. Kemperman "The stationary behavior of ideal TCP congestion avoidance", *ftp://ftp.bellcore.com/pub/tjo/TCPWindows.ps.*

[11] *www.psc.edu/networking/tcp_friendly.html*

[12] 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.

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

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

[15] 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.

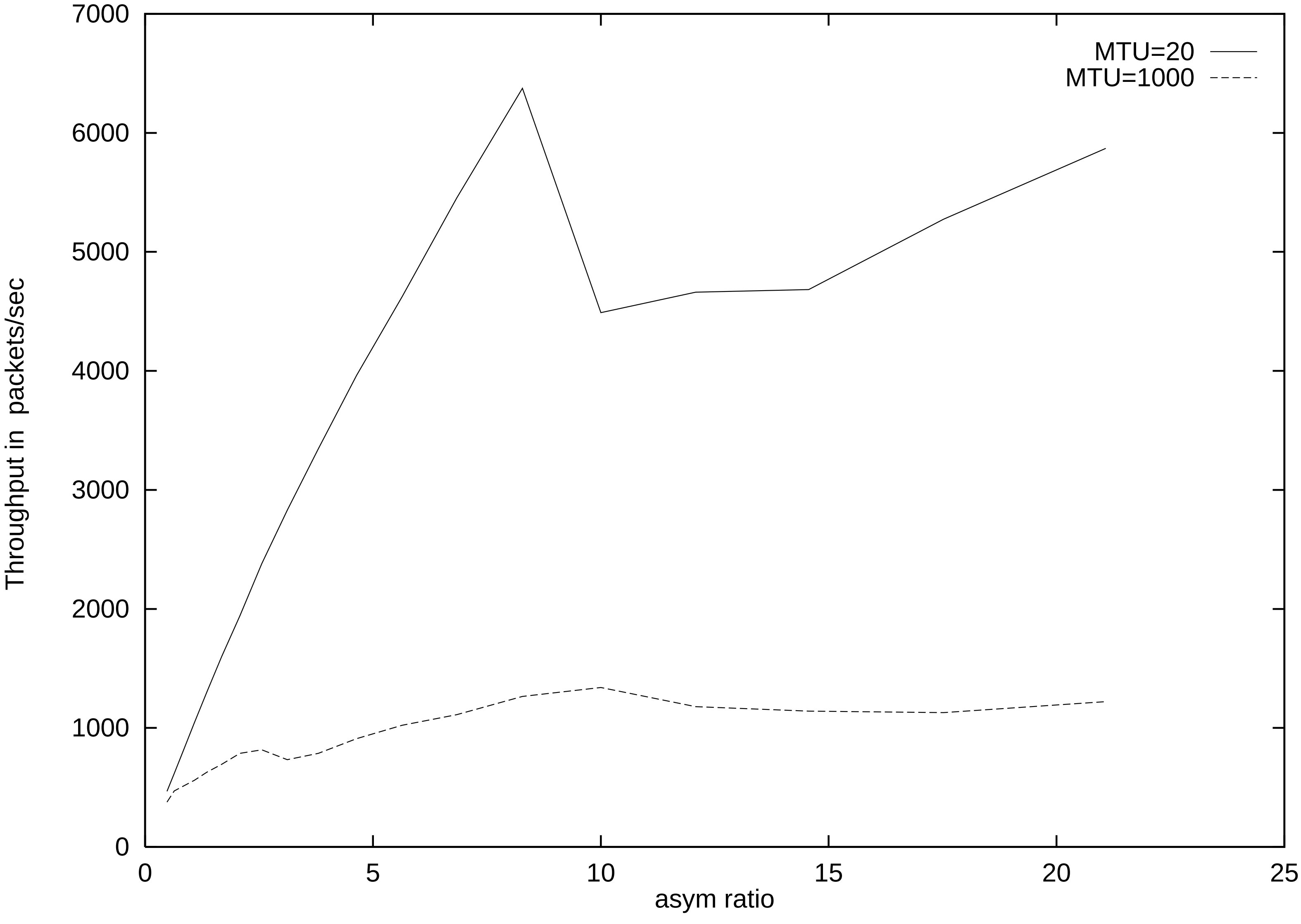

Figure 14: Effect of MTU size on throughput as a function of asymmetry, ($\mu_r = 320\ kbps$, $\tau_f = \tau_r = 2ms$, $B_f = 15$, $B_r = 5$).

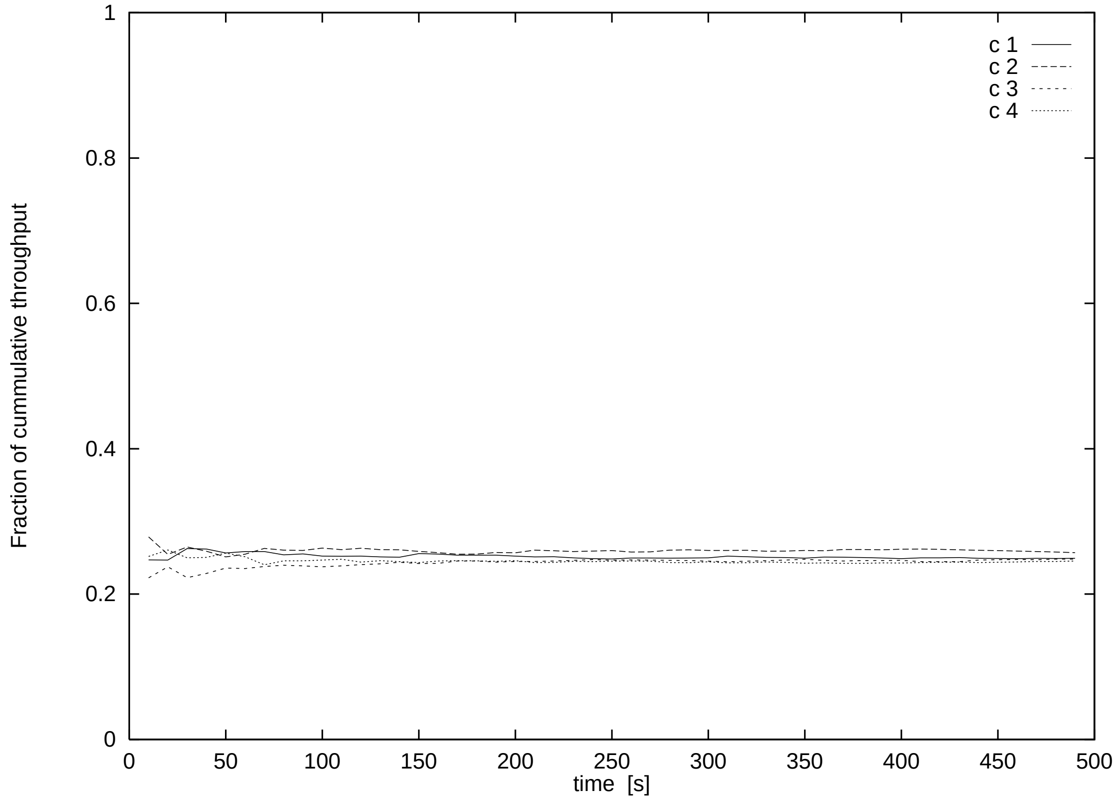

Figure 15: 4 forward connections with equal propagation delays and per connection upstream buffer ($\mu_r = 1000$ pkt/s, $k = 5$, $\tau_f = \tau_r = 1ms$, $B_f = 10$, $B_r = 5$).

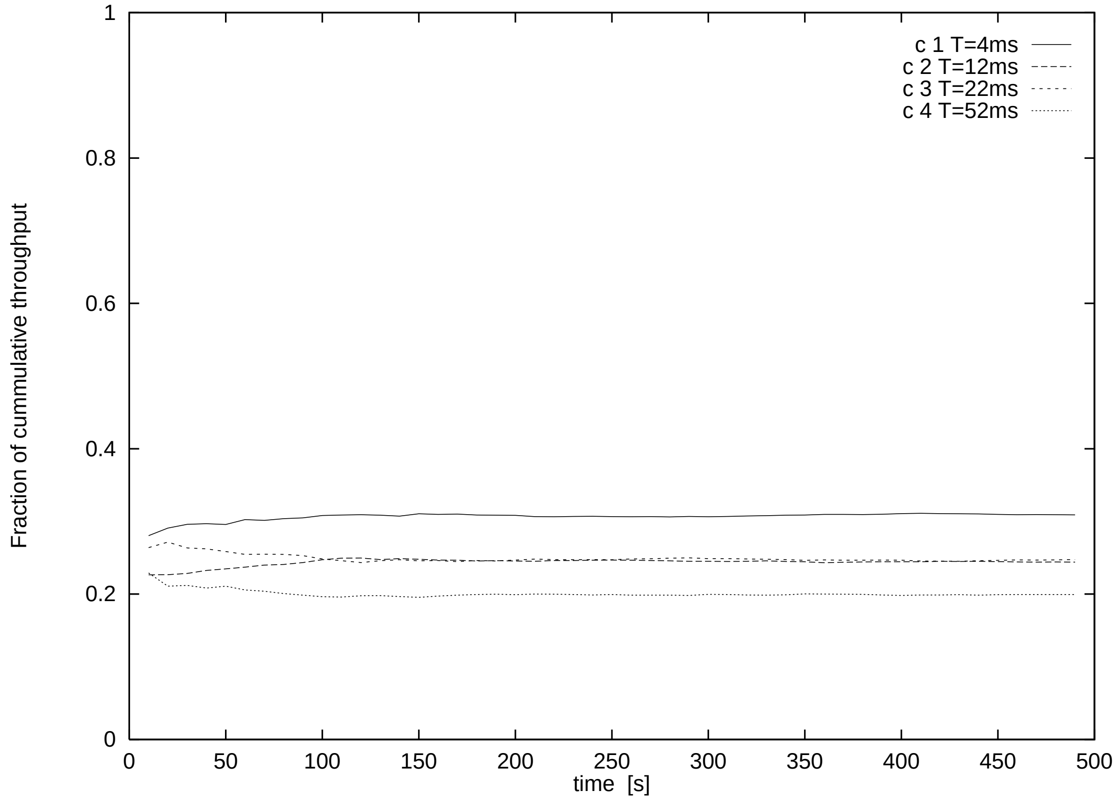

Figure 16: 4 forward connections with different propagation delays and per connection upstream buffer ($\mu_r = 1000$ pkt/s, $k = 5$, $B_f = 10$, $B_r = 5$).