Understanding the Performance of Many TCP Flows

Lili Qiu	Yin Zhang	Srinivasan Keshav
liliq@microsoft.com	yzhang@cs.cornell.edu	skeshav@ensim.com
Microsoft Research	Cornell University	Ensim Corporation

Abstract

As the most widely used reliable transport in today's Internet, TCP has been extensively studied in the past decade. However, previous research usually only considers a small or medium number of concurrent TCP connections. The TCP behavior under many competing TCP flows has not been sufficiently explored.

In this paper [1], we use extensive simulations to systematically investigate the performance of a large number of concurrent TCP flows. We start with a simple scenario, in which all the connections have the same round-trip time, and the gateways use Drop-Tail policy. We examine how the aggregate throughput, goodput, and loss rate vary with different underlying topologies. We also look at the behavior of each individual connection when competing with other connections. We observe global synchronization in some cases. We break the synchronization by either adding random processing time or using RED gateways, and examine their effects on the TCP performance. Finally we investigate the TCP performance with different RTT's, and quantify the roundtrip bias using both analysis and simulations.

Keywords: TCP, congestion control, drop-tail, RED, simulation.

1. Introduction

TCP [24] is the most widely used reliable transport protocol today. It is used to carry a significant amount of Internet traffic, including World Wide Web (HTTP), file transfer (FTP), email (SMTP) and remote access (Telnet) traffic. Due to its importance, TCP has been extensively studied in the past ten years.

There are three commonly used approaches to study the TCP protocol: developing analytical models, conducting Internet measurements, and running simulations. This paper focuses on using simulations to study the performance of *long-lived* TCP flows[2]. It extends the previous results in several important ways. Most notably, we concentrate on TCP behavior under many competing TCP flows, while previous research usually only considered a small or medium number of concurrent TCP connections. We start with a simple scenario, in which all the connections have the same round-trip time, and the gateways use Drop-Tail policy. We examine how the aggregate throughput, goodput, and loss rate vary with different underlying topologies. We also look at the behavior of each individual connection when competing with other connections. In some cases, we observe global synchronization. To break the synchronization, we either add random processing time, or use RED gateways, and examine their effects on the TCP performance. Finally we investigate the TCP performance with different RTT's, and quantify the roundtrip bias using both analysis and simulations.

The remainder of the paper is organized as follows. In Section 2, we describe our simulation methodology. In Section 3, we present the simulation results for TCP flows with the same propagation delay under Drop-Tail gateways. In Section 4 and Section 5, we study the effect of adding random overhead and using RED gateway on TCP performance, respectively. In Section 6, we examine the performance of TCP/SACK (TCP with support for *selective acknowledgments*) [30] under both Drop-Tail and RED gateways. We quantify the roundtrip bias using both simulations and analysis in Section 7. We end with concluding remarks and future work in Section 9.

2. Simulation Methodology

We study the aggregate TCP performance through extensive simulations using `ns` network simulator [33]. The `ns` Reno code closely models the congestion control behavior of most of the TCP implementations in widespread use. We use a single-bottleneck topology [3] shown in Figure 1 as

[1] This paper is a significantly modified and extended version of an earlier paper that was published in the Proceedings of the 7th International Conference on Network Protocols (ICNP'99), Toronto, Canada, 1999.

[2] A long-lived TCP flow is a TCP flow whose sender has unlimited amount of data to send.

[3] It is possible to have multiple bottlenecks. However this paper focuses on a thorough examination of single-bottleneck topologies.

our simulation topology, where the high bandwidth link is set to 10 Mbps Ethernet bandwidth with 0.1 ms delay. We vary each of the four parameters in the model: $propagation\ delay$, $BufferSize_S$, $Conn$, and $Type_{accesslink}$ to see how each parameter affects TCP performance. More specifically, we consider both ISDN and T1 links, with the link delay of either 50 ms (typical for terrestrial WAN links) or 200 ms (typical for geostationary satellite links) [37]. We also vary the buffer size and the number of connections for each scenario [4].

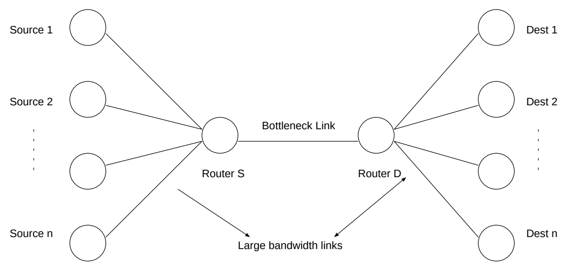


Figure 1. Simplified abstract network model.

The bottleneck link router uses FIFO scheduling. Its buffer management is either Drop-Tail or RED [16]. The TCP segment size is set to 500 bytes, similar to what was used in [32, 2]. As [32] points out, it is very common to have hundreds of concurrent connections competing for the bottleneck resource in today's Internet, so we are particularly interested in the TCP behavior under such a large number of connections.

Each sender has unlimited amount of data to send. When the bottleneck link is ISDN, the TCP sources send data with start time uniformly distributed between 0 and 5 seconds, and the duration of each simulation is 1000 seconds. When the bottleneck link is T1, the start time is uniformly distributed between 0 and 1 second, and each simulation lasts 200 seconds.

We use the following notations throughout our discussions:

- Let $W_{opt} = propagation\ delay * bottleneck\ bandwidth$, which is the number of packets the link can hold. W_{opt} is optimal in the sense that it is the window size that fully utilizes the link bandwidth without incurring any queuing delay.
- Let $W_c = W_{opt} + B$, where B is the buffer size at the bottleneck link. W_c is the total number of packets that the link and the buffer can hold.

[4] Note that to fully understand the performance implication of different parameters in network topologies, our simulation settings are not limited to the real-world scenarios.

3. TCP behavior for flows with the same propagation delay

Our study of TCP flows with the same propagation delay shows that TCP exhibits a wide range of behavior depending on the value of $\frac{W_c}{Conn}$, where $Conn$ denotes the number of connections. Based on the capacity of the pipe (measured as $\frac{W_c}{Conn}$), we classify our results into the following three cases: large pipe ($W_c > 3 * Conn$), small pipe ($W_c < Conn$), and medium pipe ($Conn < W_c < 3 * Conn$).

3.1. Case 1: $W_c > 3 * Conn$ (Large pipe case)

Previous studies have shown that a small number of TCP connections with the same RTT can get synchronized [44]. We originally thought adding more connections introduces more randomness, and makes synchronization harder to take place. Surprisingly, our simulation results show the synchronization persists even in the case of large number of connections.

Figure 2 depicts the synchronization behavior. **In all graphs we sort the connection ID's by the total number of packets each connection has received.** Such sorting allows us to reveal the synchronization behavior more clearly. As shown in the figure, the buffer occupancy periodically fluctuates from half to full, which implies all connections halve their congestion windows in synchrony. The global synchronization behavior can be further demonstrated by the periodic white stripes in the scatter plot of ACK arrival time, which imply all the connections start and end loss recovery in a synchronized manner.

The explanation for the synchronization behavior is similar to the case for a small number of connections. Suppose at the end of the current epoch the total number of outstanding packets from all the connections is equal to W_c. During the next epoch all connections will increment their window. All the packets that are sent due to window increase will get dropped. Thus all the connections will incur loss during the same RTT. This makes all connections adjust window in synchrony. When $W_c > 3 * conn$, most connections have more than 3 outstanding packets before the loss. So they can all recover the loss by fast retransmissions, and reduce the window by half, leading to global synchronization. In contrast, when $W_c < 3 * Conn$, though all the connections still experience loss during the same RTT, they react to the loss differently. Some connections whose $cwnd$ is larger than 3 before the loss can recover the loss through fast retransmission, whereas the others will have to use timeout to recover the loss. Since the set of connections recovering loss using fast transmission and the set using timeout will change over time, global synchronization cannot be achieved.

Due to global synchronization, all the connections share the resource very fairly: in the steady state they experience

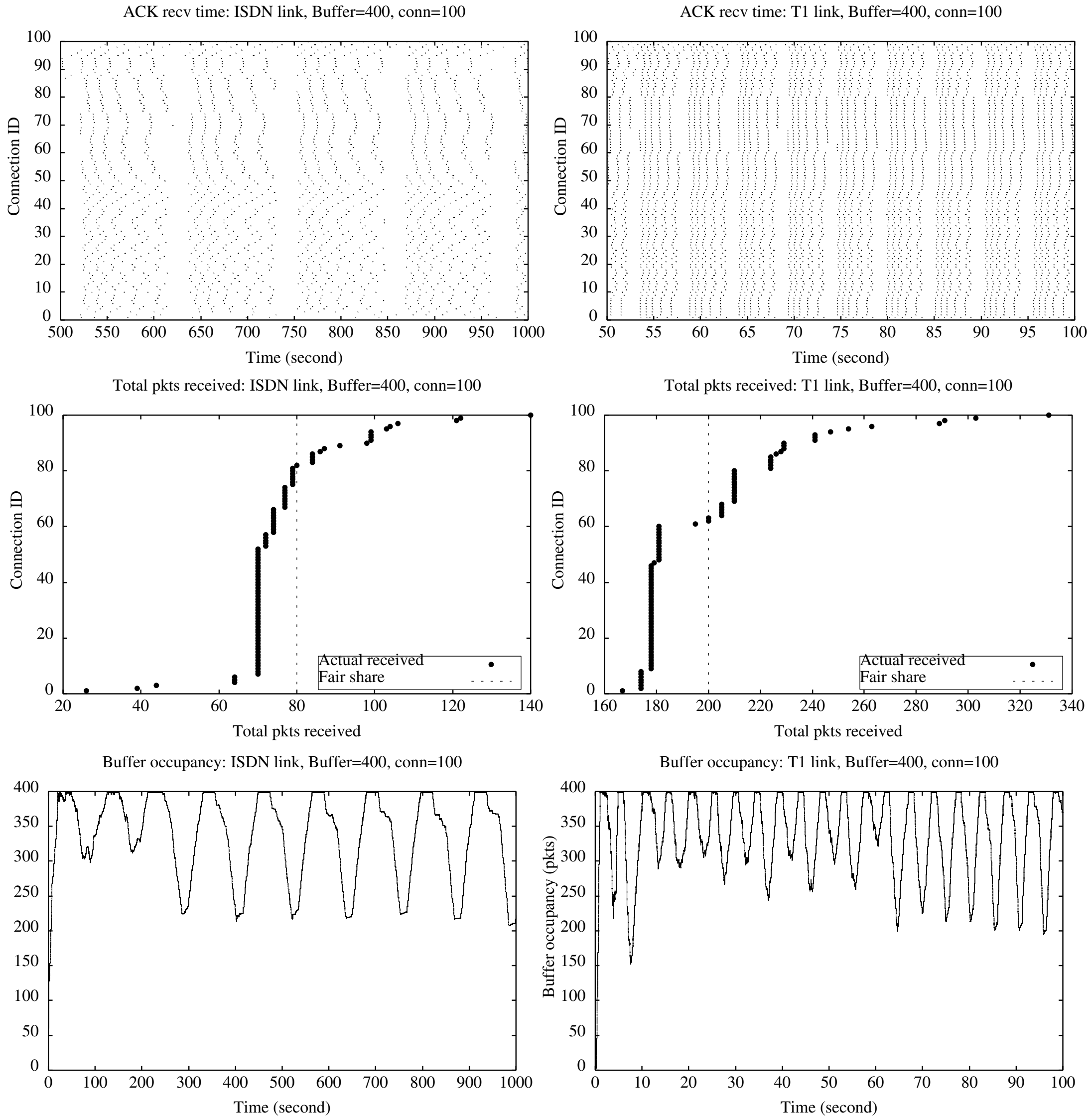


Figure 2. Large pipe case leads to global synchronization: 100 connections sharing the bottleneck link of either ISDN or T1 with one-way propagation delay of 50 ms and bottleneck buffer size of 400 packets. Notice the buffer occupancy fluctuates from half to full periodically. Moreover there are periodic white stripes in the scatter plot of ACK arrival time, where during the white stripes all connections experience loss and try to recover them using fast retransmission.

the same number of losses and send the same number of packets. We can aggregate all the connections as one big connection, and accurately predict the aggregate loss probability as follows:

$$Loss\ Probability = \frac{1}{b * (\sum_{x=\lfloor \frac{W+1}{2} \rfloor}^{W} x) + 2 * W + 1}$$

where b is the average number of packets acknowledged by an ACK, and $W = \frac{W_c}{Conn}$. (Refer to [41] for more details.) When $b = 1$,

$$Loss\ Probability = \begin{cases} \frac{8}{3*W^2+20*W+9} & \text{if W is odd} \\ \frac{8}{3*W^2+22*W+8} & \text{otherwise} \end{cases}$$

So we can approximate $Loss\ Probability$ as

$$Loss\ Probability \approx \frac{8}{3 * W^2 + 21 * W + 8}.$$

Figure 3 shows our prediction matches well to the actual loss probability.

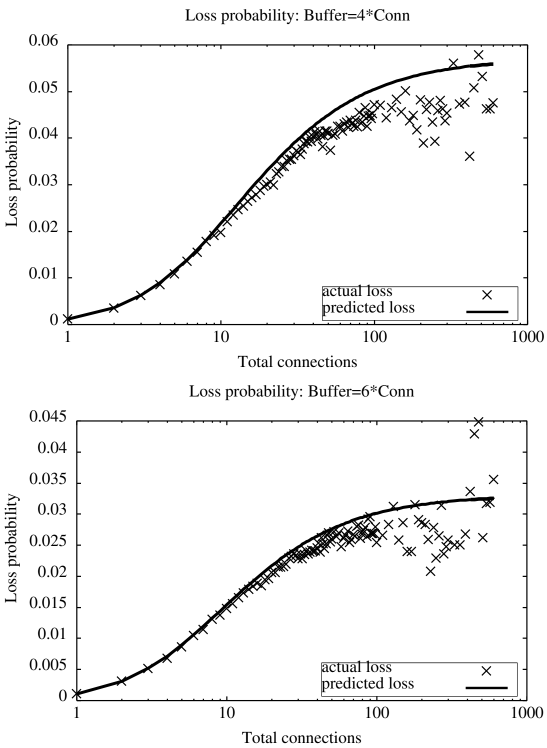


Figure 3. Loss prediction for large pipe case: A varying number of connections share T1 link with one-way propagation delay of 50 ms, and the bottleneck buffer is either 4 times or 6 times the number of connections.

Furthermore, as expected, global synchronization leads to periodical fluctuation in buffer occupancy as shown in Figure 2. When the total buffer size is less than $\frac{W_c}{2}$, halving $cwnd$ in synchrony leads to under-utilization of bottleneck link.

3.2. Case 2: $W_c < Conn$ (Small pipe case)

When $W_c < Conn$, we have found TCP connections share the path very unfairly: only a subset of connections are *active* (i.e. their goodput is considerably larger than 0), while the other connections are shut-off due to constant timeout as shown in Figure 4. The number of active connections is close to W_c, and the exact value depends on both W_c and the number of competing connections. When $Conn$ exceeds the number of active connections the network resource can support, adding more connections to the already overly congested network only creates more shut-off connections. Almost all the packets sent by the shut-off connections get dropped. The remaining active connections are left mostly intact. This explains the curves in Figure 5: when $Conn$ is larger than the number of active connection the network resources can support, the total number of packets sent and lost grows linearly with the number of connections. The linear increase in the number of packets sent and lost mainly comes from the increase in the number of inactive (mostly shut-off) connections, each of which sends a constant number of packets before it gets completely shut off.

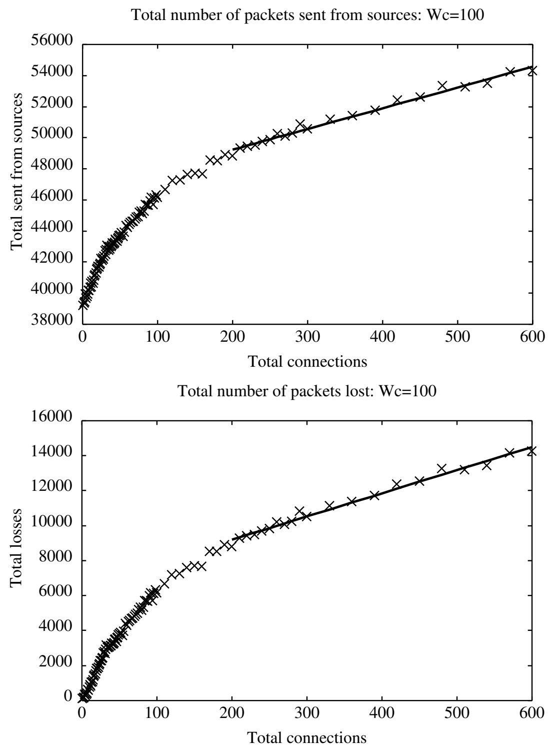


Figure 5. Varying number of connections compete for T1 link with $delay = 50\ ms$ and $buffer = 60\ packets$ ($W_c = 100\ packets$): the total number of packets sent and dropped increases linearly with the number of connection when the connection number is large.

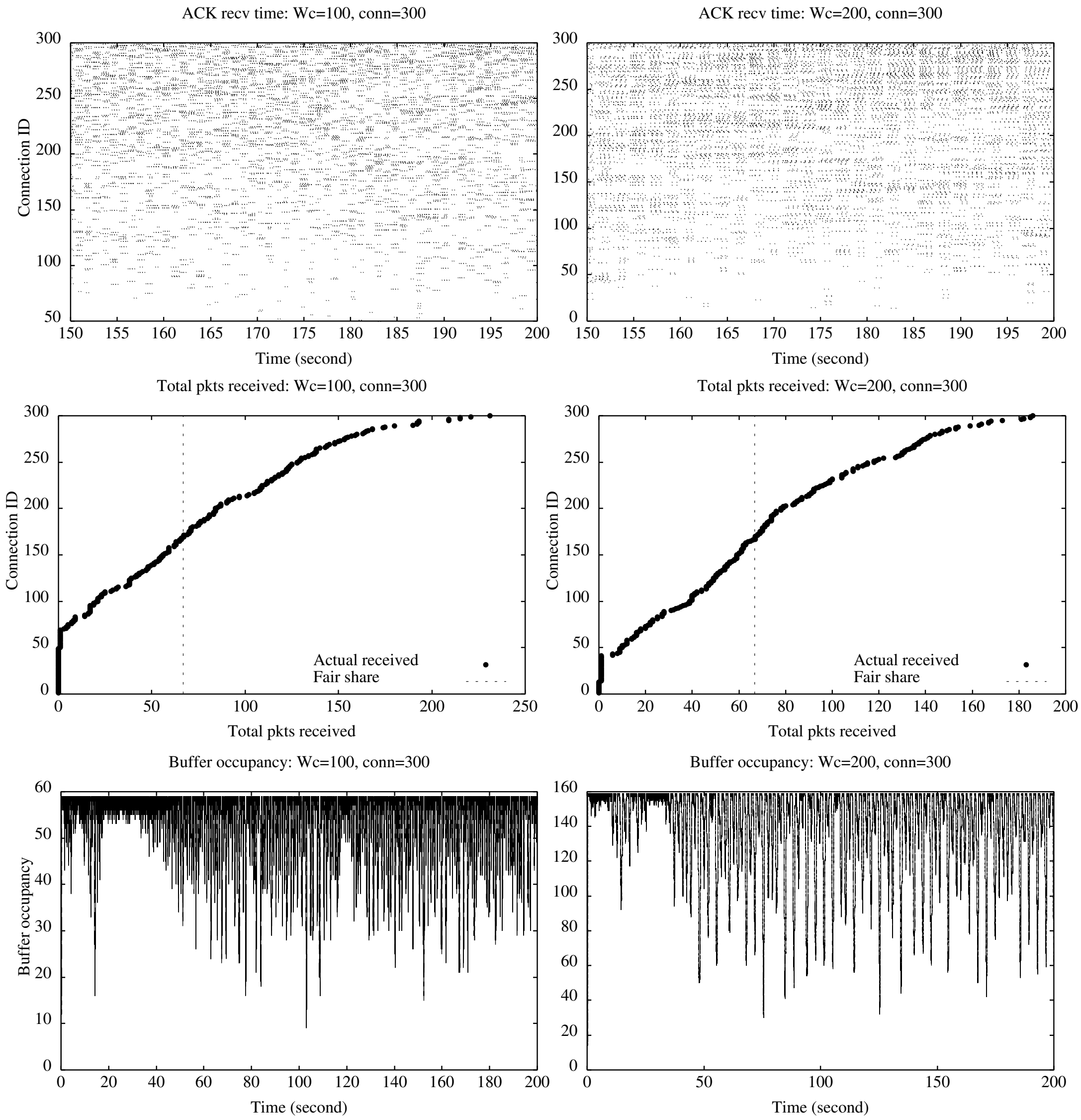


Figure 4. Small pipe case: 300 concurrent connections compete for T1 link with one-way propagation delay of 50 ms and bottleneck buffer size of 60 (or 160) packets ($W_c = W_{opt} + Buffer =$ 100 or 200 packets). Note that the buffer occupancy fluctuates widely. Moreover part of connections receive little goodput as shown in the two graphs in the middle.

3.3. Case 3: $Conn < W_c < 3 * Conn$ (Medium pipe case)

As shown in Figure 6, TCP behavior in this case falls in between the above two cases. More specifically, as explained earlier (in Section 3.1), since $1 < \frac{W_c}{Conn} < 3$, the connections respond to loss differently: connections with $cwnd$ greater than 3 before the loss can recover from the loss through fast retransmission, whereas the others will have to recover from the loss using timeout. Since the set of connections using fast retransmission and the set using timeout will change over time, no global synchronization occurs, and the network resources are not shared as fairly as when $W_c > 3 * Conn$. On the other hand, there is still local synchronization, as shown Figure 6, where some groups of connections are synchronized within the groups. Furthermore, since $\frac{W_c}{Conn} > 1$, all the connections can get reasonable amount of throughput. In contrast to the small pipe case, there are almost no connections getting shut-off.

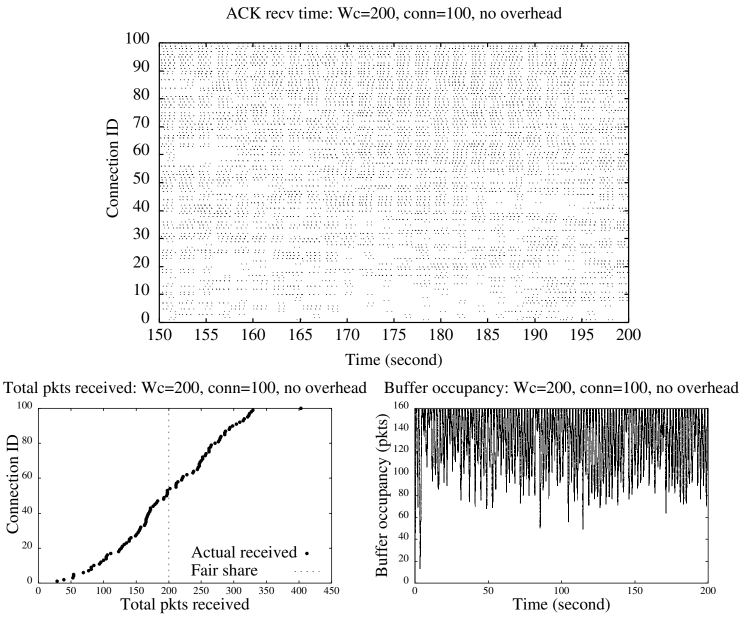
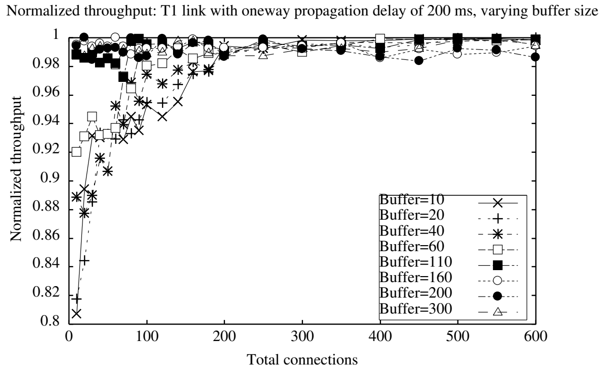


Figure 6. Medium pipe case: 100 concurrent connections compete for T1 link with $delay = 50\ ms$ and $buffer = 160\ packets$ ($W_c = W_{opt} + Buffer = 200\ packets$). Buffer occupancy fluctuates widely, and there is local synchronization within some groups.

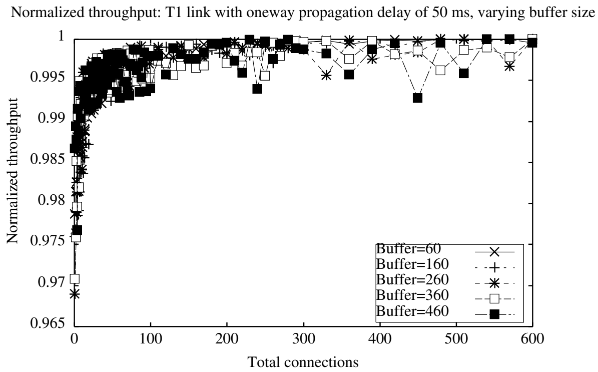
3.4. Aggregate Throughput

We define *normalized aggregate TCP throughput* as the number of bits sent by the bottleneck link in unit time normalized by the link capacity. Our results are as follows:

- As shown in Figure 7(a), when the number of connections is small and the buffer size is less than W_{opt} ($W_{opt} = 160$ packets in this case), the normalized TCP throughput is less than 1. The degree of under-utilization depends on both the number of connections and the ratio of the buffer size to W_{opt}. The smaller the number of connections and the lower the ratio, the lower the network utilization is.
- As shown in Figure 7(b), when the buffer size is larger than W_{opt} ($W_{opt} = 40$ packets in this case), the normalized TCP throughput is close to 1, regardless of the number of connections.
- When the number of connections is large, even if the buffer size is small (smaller than W_{opt}), the normalized TCP throughput is close to 1. This is evident from Figure 7(a), where the throughput is close to 1 for a large number of connections under all the buffer sizes considered.



(a)



(b)

Figure 7. Throughput: varying number of connections compete for the bottleneck link T1 with $delay = 200\ ms$ or $delay = 50\ ms$.

3.5. Aggregate Goodput

We define *normalized aggregate goodput* as the number of *good* bits, i.e. bits that are not unnecessarily retransmitted, received by all the receivers in unit time normalized by the link capacity. As shown in Figure 8,

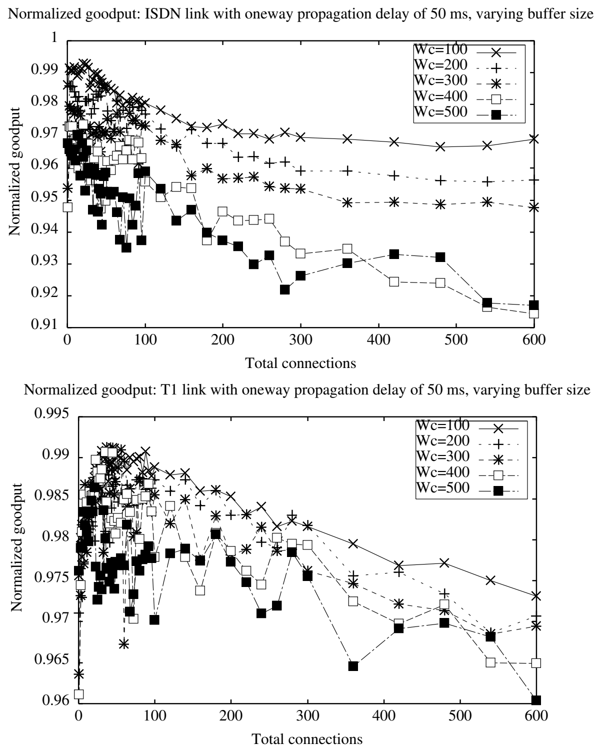


Figure 8. Goodput: varying number of connections compete for the bottleneck link of either ISDN or T1. In both cases, $delay = 50\,ms$.

- There is a linear decrease in goodput as the number of connections increases.
- The slope of the decrease depends on the bottleneck link bandwidth: the decrease is more rapid when the bottleneck link is ISDN, and is slower when T1 is used as the bottleneck link.

We can explain these results as follows. The difference between the throughput and goodput is the number of unnecessary retransmissions. As the number of connections increases, the loss probability increases, which in turn increases the number of unnecessary retransmissions. Therefore the more connections, the lower the goodput is. On the other hand, since

$$\begin{aligned} & loss\ in\ the\ normalized\ goodput \\ = & \frac{total\ unnecessary\ retransmissions}{link\ capacity} \end{aligned}$$

the decrease in the goodput is more significant with slower bottleneck link (e.g. ISDN), and less significant with faster bottleneck link (e.g. T1). This is evident from Figure 8.

To summarize, when the bottleneck link is fast or the loss probability is low (less than 20%), the number of unnecessary retransmissions is negligible. So the normalized goodput is close to the normalized throughput. Otherwise (i.e., when the bottleneck link is slow and the loss probability is high) the loss in the goodput due to unnecessary retransmissions becomes significant. The decrease in the goodput depends on both the bottleneck link bandwidth and the loss probability.

3.6. Loss Probability

Our simulation results indicate when the W_c is fixed and the number of connections is small, the loss probability grows quadratically with the increasing number of connections as shown in Figure 9. The quadratic growth in the loss probability can be explained as follows. When $\frac{W_c}{Conn} > 3$, TCP connections can recover packet loss without timeouts. Every connection loses one packet during each loss episode. So altogether there are $Conn$ losses every episode. Meanwhile the frequency of loss episode is proportional to $Conn$. Therefore for a small number of connections, the loss probability is proportional to $Conn^2$. Such quadratic growth in loss probability is also reported in [32] for TCP/Tahoe with RED dropping policy.

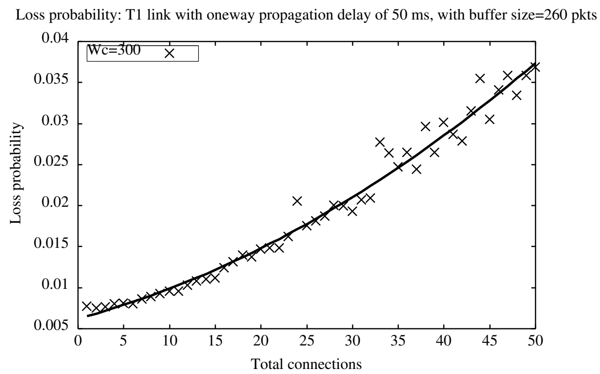


Figure 9. Loss probability for <u>a small number of connections</u>: varying number of connections compete for the bottleneck link of T1 with $delay = 50\,ms$**. The loss probability grows quadratically when the number of connections is small.**

As the number of connections gets large (larger than $\frac{W_c}{3}$), the growth of loss probability with respect to the number of connections matches very well with the following family of hyperbolic curves represented by $y = \frac{b*x}{x+a}$ as shown in Figure 10. Table 1 gives the parameters of the hyperbolic curves used in Figure 10.

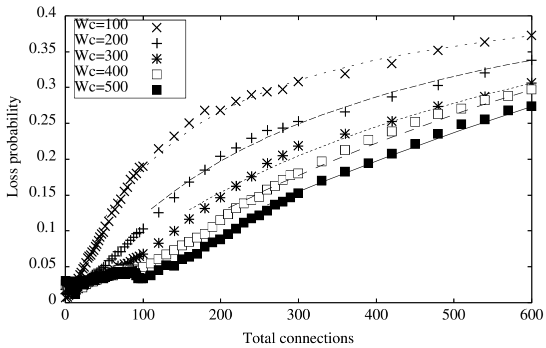
A possible explanation for the hyperbolic growth in loss rate is as follows. When each connection's share of W_c is small, most connections stay in slow start. Assuming sufficient statistical multiplexing, every connection's congestion window grows from 1 to W, so the total number of packets sent in one epoch is $2 * W + c$, where $W \propto \frac{W_c}{Conn}$, and c is a small constant corresponding to the number of packets sent during consecutive timeouts [5]. Since the loss rate is inversely proportional to the total number of packets sent in

[5] Assuming there is no delayed ACK, and a connection encounters a

one epoch, we have $loss\ rate \propto \frac{1}{2W+c}$, which is equivalent to

$$loss\ rate \propto \frac{\frac{1}{2W_c} * Conn}{1 + \frac{c}{2W_c} * Conn}$$

This conforms to the hyperbolic function.



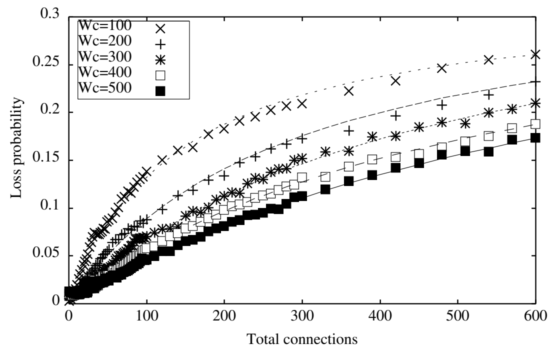


Figure 10. Loss probability for a large number of connections: varying number of connections compete for the bottleneck link of either ISDN or T1. In both cases, propagation delay = 50 ms. The loss probability curves match very well with hyperbolic curves when the number of connections is large, where the parameters of the hyperbolic curves are given in Table 1.

ISDN			T1		
Wc	a	b	Wc	a	b
100	149.2537	0.4646	100	144.9275	0.3237
200	333.3333	0.5262	200	285.7143	0.3429
300	588.2352	0.6059	300	454.5454	0.3682
400	1250.000	0.9170	400	526.3158	0.3517
500	2000.000	1.1852	500	769.2308	0.3948

Table 1. Parameters for the hyperbolic curves used for fitting loss probability as shown in Figure 10.

4. TCP behavior with random overhead

4.1. $W_c > 3 * Conn$ (Large pipe case)

Our discussions in Section 3 focus on the macroscopic behavior of concurrent TCP connections with the same propagation delay. In order to explore properties of networks with Drop-Tail gateways unmasked by the specific details of traffic phase effects or other deterministic behavior, we add random packet-processing time in the source nodes. This is likely to be more realistic. The technique of adding random processing time was first introduced in [15]. However we have different goals. In [15], Floyd and Jacobson are interested in how much randomness is necessary to break the systematic discrimination against a particular connection. In contrast, we are interested in how much randomness is sufficient to break the global synchronization. Consequently, the conclusions are different. [15] concludes that adding a random packet-processing time ranging from zero to the bottleneck service time is sufficient; while we find that a random processing time that ranges from zero to $10\% * RTT$ (usually much larger than a random packet service time) is required to break down the global synchronization. This is shown in Figure 11, where the global synchronization is muted after adding the random processing time up to $10\% * RTT$.

The performance results in the non-synchronization case also differ from the global synchronization case. As shown in Figure 12, when the number of connections is less than 100, the loss probability in the two cases are almost identical; as the number of connections further increases, the gap between the two opens up—the non-synchronization case has higher loss probability than the synchronization case. Nevertheless, using the prediction based on global synchronization gives a reasonable approximation (at least a lower bound) of loss probability for non-synchronized case. However, in the non-synchronization case, the connections do not share the bottleneck bandwidth fairly. As shown in Figure 11, there is a large variation in the throughput of different connections. Consequently, we can no longer predict the bandwidth share for each connection in this case.

A few comments follow:

- There is a general concern that synchronization is harmful since it may lead to under-utilization of the bottleneck bandwidth. However with the use of TCP/Reno and sufficient bottleneck buffer provisioning, this is unlikely to be a problem in practice.
- There have been several attempts to break synchronization in order to avoid under-utilization. However our simulation results indicate breaking the synchronization by adding random processing time increases the unfairness and loss probability.

packet drop (or drops) during the slow start phase when the congestion window is W. Then the total number of packets sent in one epoch before a timeout is $1 + 2 + ... + \frac{W}{2} + W = 2W - 1$.

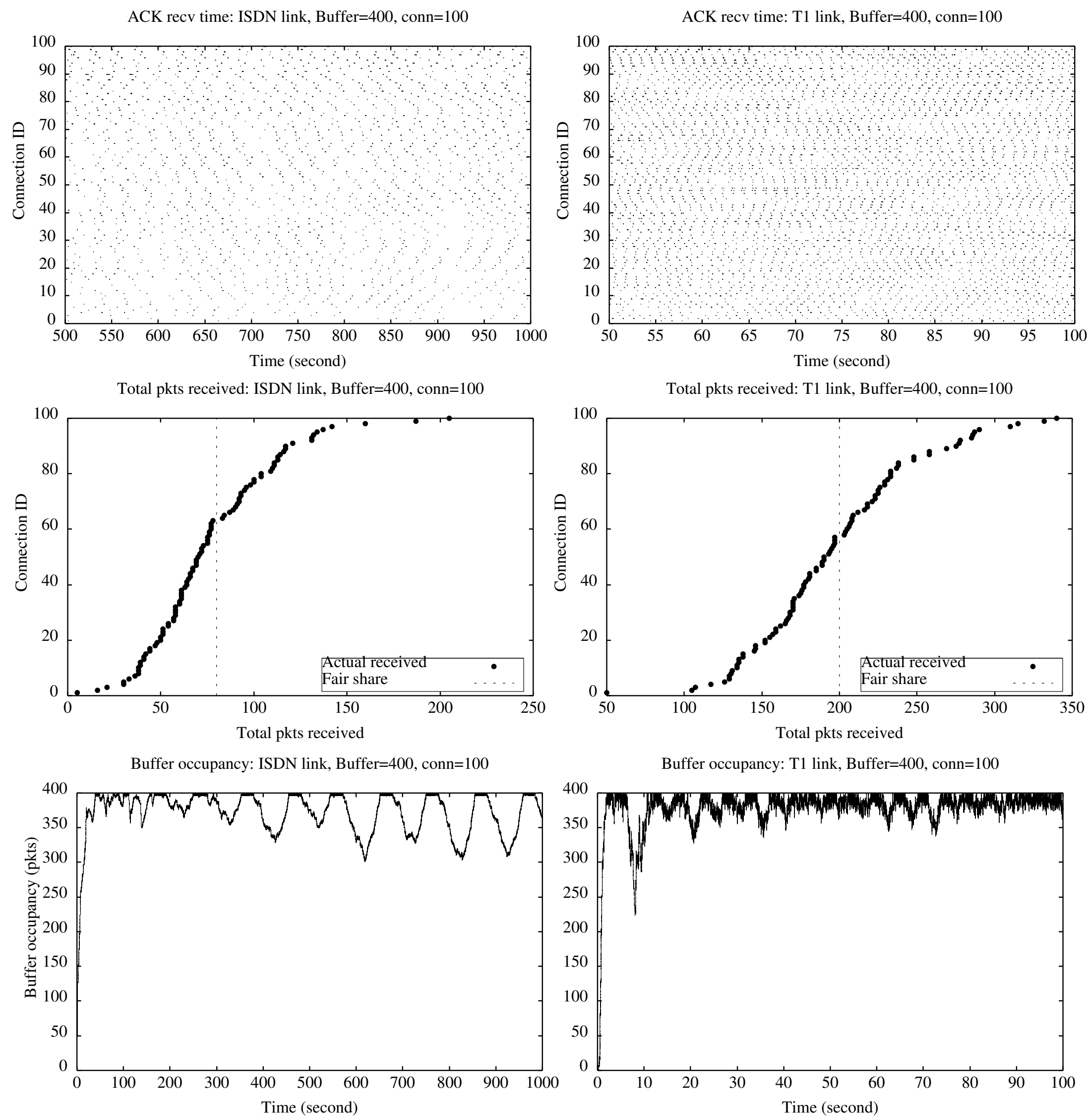


Figure 11. Adding random process time in large pipe case breaks down the global synchronization: 100 connections share the bottleneck link of either ISDN or T1 with one-way propagation delay of 50 ms and bottleneck buffer size of 400 packets. Compared to the case of without random processing time, the buffer occupancy is quite stable. Furthermore global synchronization disappears as shown in the scatter plot for ACK arrival.

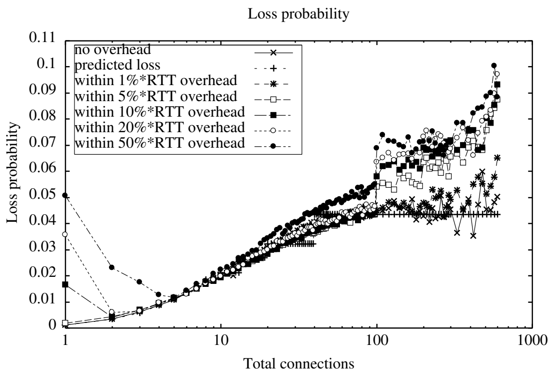


Figure 12. Compare the loss probability by adding different amount of random processing time: varying number of connections compete for T1 link with $delay = 50\,ms$.

4.2. $W_c < Conn$ (Small pipe case)

Adding random processing time also affects the case when $W_c < Conn$. Without random processing time, we find there is consistent discrimination against some connections, which end up being completely shut off due to constant timeouts. After adding random processing time, such discrimination still exists but is much less severe. As shown in Figure 13, the number of shut-off connections is considerably smaller than before. Furthermore, the buffer occupancy is mostly full and stable, whereas without random processing time, the buffer occupancy is quite low, and fluctuates a lot.

4.3. $Conn < W_c < 3 * Conn$ (Medium pipe case)

As shown in Figure 14, adding random processing time does not have much impact on TCP behavior in the case of medium size pipe: as before, most connections get reasonable goodput and are not synchronized. On the other hand, the buffer occupancy now becomes mostly full and stable in contrast to the case without random processing time, where the buffer occupancy is low, and fluctuates a lot. In addition, even local synchronization disappears after adding random processing time.

4.4. Aggregate Throughput & Goodput

Adding random processing time changes little in the overall throughput and goodput as shown in Figure 15 and Figure 16.

4.5. Loss Probability

With a small number of TCP connections, after adding random processing time, the loss probability grows mostly

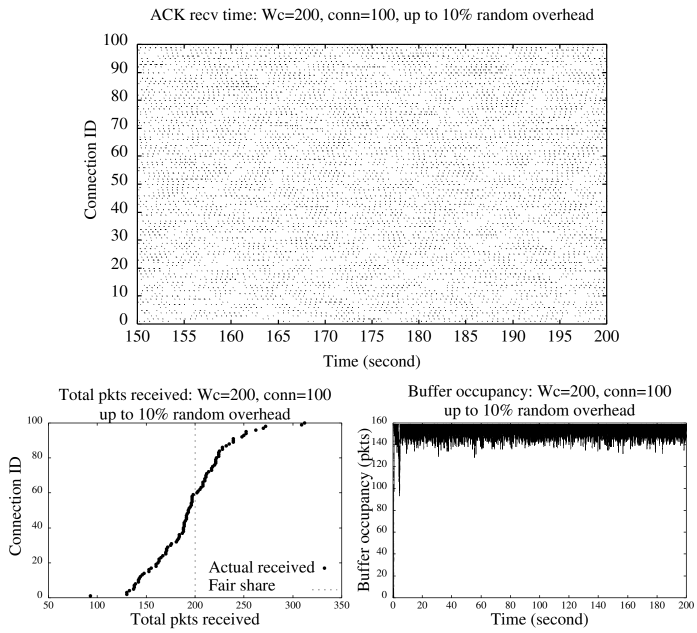


Figure 14. Adding random processing time in medium pipe case: 100 concurrent connections compete for T1 link with $delay = 50\,ms$ and $buffer = 160\,packets$ ($W_c = W_{opt} + Buffer = 200$ packets). The buffer occupancy is quite stable. In contrast to without random processing time, there is no local synchronization.

linearly as the number of connections increases. This is evident from Figure 17, which compares the loss probability curves before and after adding random processing time.

For the large number of connections, adding random processing time does not change the general shape of the loss probability curve: as before, the growth of loss probability with respect to the number of connections matches well with hyperbolic curves, as shown in Figure 18. On the other hand, for the same configuration (with the same number of connections and buffer size), the loss probability becomes higher after adding the random processing time. Consequently, the parameters of hyperbolic curves are different as shown in Table 2.

ISDN			T1		
Wc	a	b	Wc	a	b
100	125.0000	0.4930	100	125.0000	0.4263
200	217.3913	0.5257	200	227.2727	0.4505
300	400.0000	0.6071	300	333.3333	0.4690
400	714.2857	0.7536	400	500.0000	0.5123
500	1428.5714	1.1110	500	666.6666	0.5366

Table 2. Parameters for the hyperbolic curves used for fitting loss probability as shown in Figure 18.

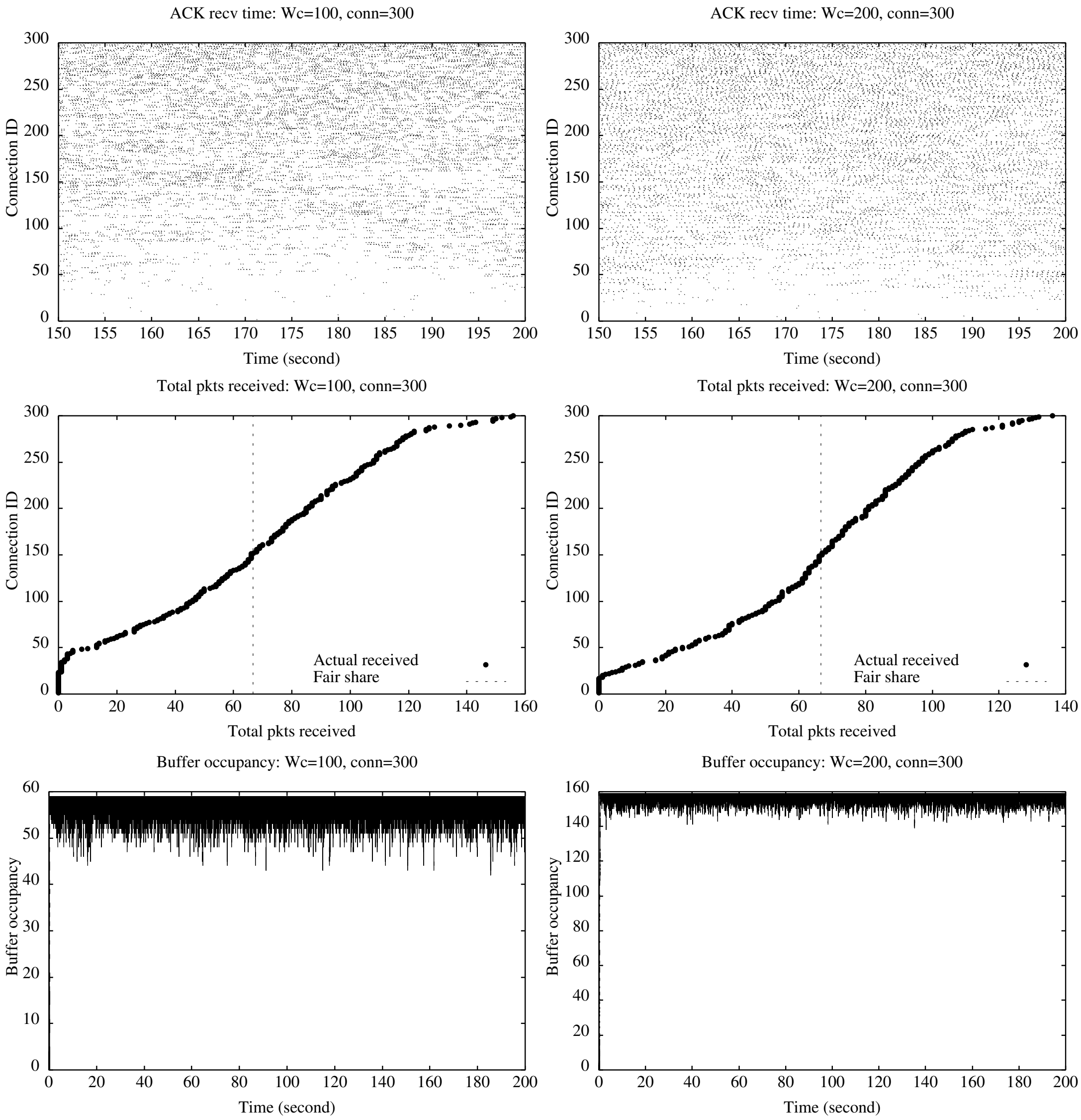


Figure 13. Adding random processing time in small pipe case: 300 concurrent connections compete for T1 link with one-way propagation delay of 50 ms and buffer size of 60 (or 160) packets ($W_c = W_{opt} + Buffer$=100 or 200 packets). The buffer occupancy is quite stable, and the consistent discrimination is not as severe as without adding random processing time.

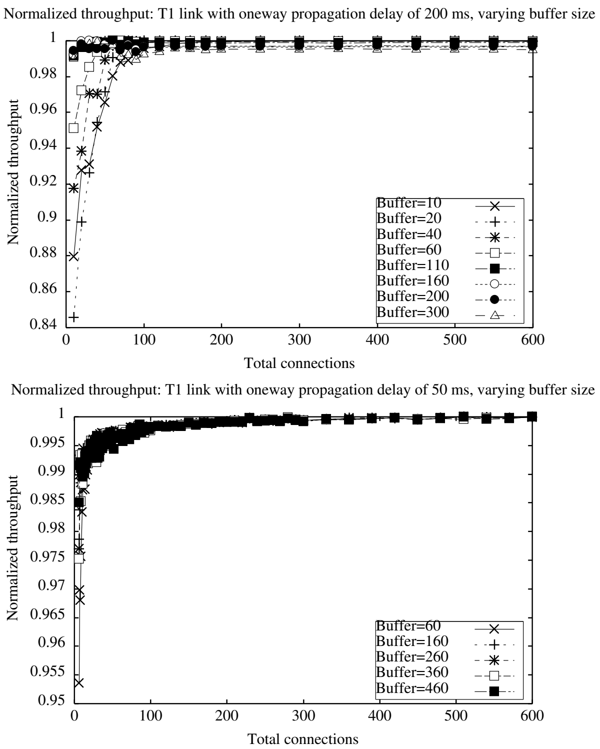


Figure 15. Throughput after adding random processing time of up to $10\% * RTT$ at TCP source: varying number of connections compete for the bottleneck link of T1 link with propagation delay of either 200 ms or 50 ms.

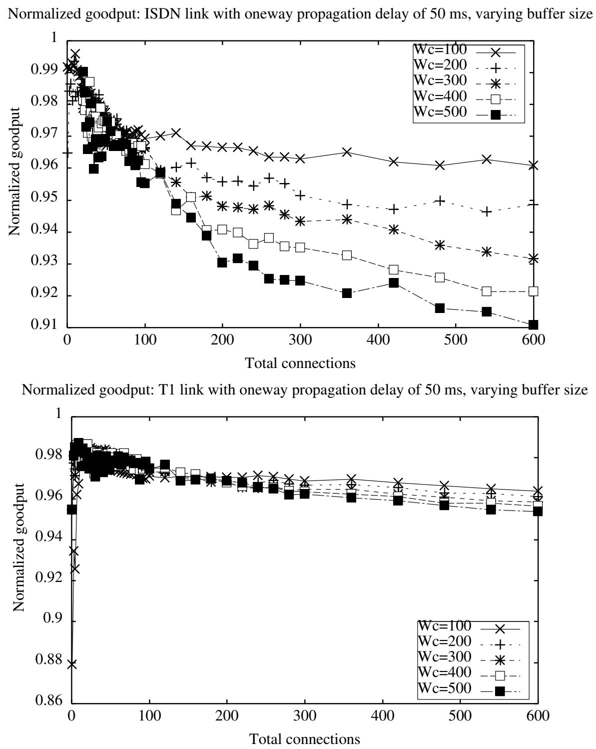


Figure 16. Goodput after adding random processing time of up to $10\% * RTT$ at TCP source: varying number of connections compete for the bottleneck link of either ISDN or T1. In both cases, the one-way propagation delay=50 ms.

5. TCP Performance under RED Queue Management

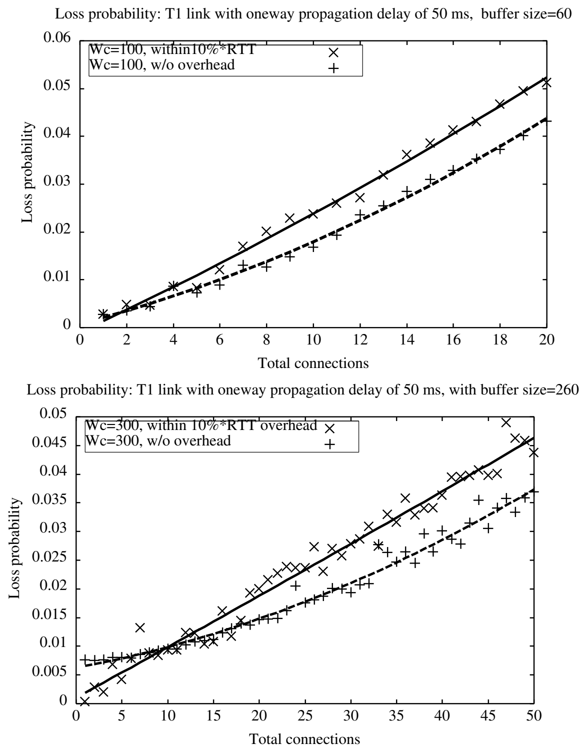
In this section, we further investigate TCP performance under Random Early Detection (RED) queue management.

5.1. Background of RED

RED is the recommended active queue management scheme for rapid deployment throughout the Internet. With RED, a router will detect congestion before the queue overflows, and provide an indication of this congestion to the end nodes. It may use one of several methods for indicating congestion to end-nodes. One is to use packet drops. Alternatively, it can set a Congestion Experienced (CE) bit in a packet header as an indication of congestion, instead of relying solely on packet drops. The latter method is commonly referred to as *explicit congestion notification* (ECN) [17, 42, 43]. The major advantage of active queue management mechanisms like RED is that the transport protocols with congestion control (e.g., TCP) do not have to rely on buffer overflow as the only indication of congestion. This can potentially reduce unnecessary queuing delay for all traffic sharing that queue.

Consider a router with a buffer size of B packets. With the RED buffer management scheme, a router detects congestion by the average queue length ($\bar{q}$), which is estimated using an exponentially weighted moving average: $\bar{q} \leftarrow (1 - w_q) \cdot \bar{q} + w_q \cdot q$, where w_q is a fixed (small) parameter and q is the instantaneous queue length. When the average queue length exceeds a minimum threshold ($\min_{th}$), incoming packets are probabilistically dropped or marked with the Congestion Experienced bit [17, 42, 43]. The probability that a packet arriving at the RED queue is either dropped or marked depends upon several control parameters of the algorithm. An initial drop/mark probability P_b is computed using a drop function D based on the average queue length $\bar{q}$ and three control parameters $\max_p$, $\min_{th}$, and $\max_{th}$. The actual probability is a function of the initial probability and a count of the number of packets enqueued since the last packet was dropped: $P_a = P_b / (1 - count \times P_b)$.

In the original RED scheme, $D(\bar{q}) = 1$ if $\bar{q} \geq \max_{th}$, which means all incoming packets are dropped or marked when the average queue length exceeds $\max_{th}$. As shown by Firoiu *et al.*[13], this can lead to oscillatory behavior. Recently, Floyd recommended using the 'gentle_" variant of RED [19], which uses the following modified dropping function D (illustrated in Fig. 19):



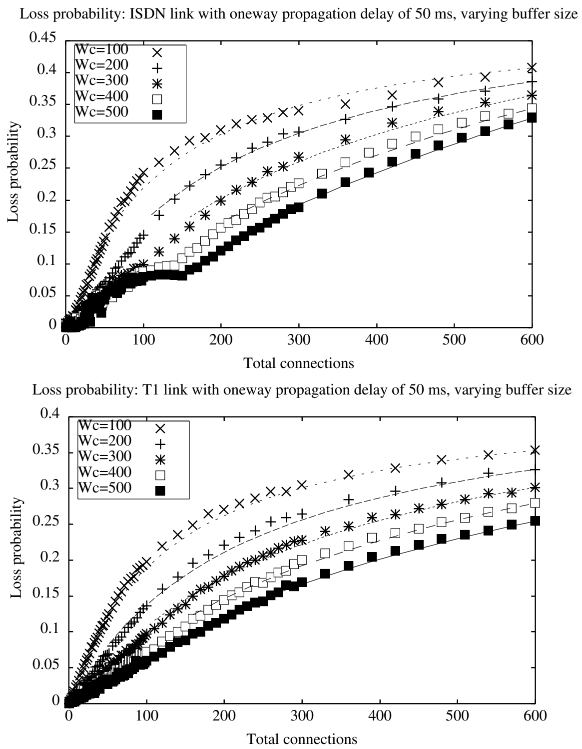


Figure 17. Loss probability for a small number of connections after adding random processing time of up to $10\% * RTT$ at TCP source: varying number of connections compete for the bottleneck link of T1 link with $delay = 50\ ms$. The loss probability grows linearly when the number of connections is small.

Figure 18. Loss probability for large number of connections after adding up to $10\% * RTT$ at TCP source: varying number of connections compete for the bottleneck link of either ISDN or T1. In both cases, the one-way propagation delay is 50 ms. The loss probability curves match very well with hyperbolic curves when the number of connections is large, where the parameters of the hyperbolic curves are given in Table 2.

$$D(\bar{q}) = \begin{cases} 0 & \text{if } \bar{q} < \min_{th} \\ 1 & \text{if } \bar{q} \geq 2 \cdot \max_{th} \\ \frac{\bar{q} - \min_{th}}{\max_{th} - \min_{th}} \cdot \max_p & \text{if } \bar{q} \in [\min_{th}, \max_{th}] \\ \frac{\bar{q} - \max_{th}}{\max_{th}} \cdot (1 - \max_p) + \max_p & \text{otherwise} \end{cases}$$

As shown by Rosolen *et al.*[39, 40], the "gentle_" option makes the RED much more robust to the setting of the parameters $\max_{th}$ and $\max_p$. Therefore, we turn it on for all simulations in this paper. Moreover, in this paper, we only consider RED without support for ECN.

5.2. RED Configurations

RED has four control parameters: $\min_{th}$, $\max_{th}$, $\max_p$, and w_q. How to properly configure these parameters has been the subject of many studies [8, 13, 11, 18]. The focus of our work is on understanding the impact of RED on TCP performance. Therefore, instead of proposing any new recommendations on configuring RED parameters, we closely follow the guidelines by Floyd [18]. More specifically, we always use the recommended values of $\max_{th} = 3\min_{th}$, $\max_p = 0.1$, and $w_q = 0.002$. The recommended value for the last parameter $\min_{th}$ is 5 packets. However, as noted in [18], the optimal setting for $\min_{th}$ also depends partly on the link speed, propagation delay, and maximum buffer size. Therefore, besides the recommended value of 5 packets, we also experiment with two different values of $\min_{th}$ based on the buffer size: $B/6$ and $B/9$, where B is the maximum buffer size. Finally, as recommended in [19], we turn on the "gentle_" option for RED in all simulations.

Table 3 lists the configurations we use for simulations.

Scheme	$\min_{th}$	$\max_{th} = 3 * \min_{th}$
RED-1	$B/6$	$B/2$
RED-2	$B/9$	$B/3$
RED-3	5 packets	15 packets

Table 3. Different RED configurations.

Under RED queue management, each flow's fair share of the pipe can be measured as $\frac{W_c^*}{Conn}$, where $Conn$ is the total number of connections, $W_c^* = W_{opt} + \max_{th}$ reflects the total number of packets the link and the buffer can hold

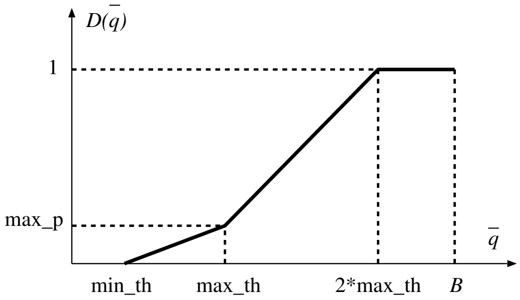


Figure 19. Drop function of RED with the “gentle_” modification.

before the router exits the operation region of RED and significantly increases the dropping probability.

Our study shows that the TCP behavior under RED queue management can again be classified into three cases based on each flow’s fair share of the pipe ($\frac{W_c^*}{Conn}$): large pipe case ($W_c^* > 3 * Conn$), small pipe case ($W_c^* < Conn$), and medium pipe case ($Conn < W_c^* < 3 * Conn$). Table 4 summarizes the TCP behavior in each case. Details are given in the following three sections.

	Pipe		
Behavior	large	small	medium
synchronization	no	no	no
link sharing	fair	unfair	fair
buffer occupancy	stable	unstable	unstable

Table 4. TCP behavior under RED queue management.

5.3. $W_c^* > 3 * Conn$ (Large pipe case)

Figure 20 shows the TCP behavior under RED queue management in the large pipe case. The randomness introduced by RED completely breaks down the global and local synchronization, which is evident from the scatter plot of the ACK arrival times. The number of packets received by different flows ranges from 120 to 280, which is quite fair. Moreover, the queue length is quite stable and is well-bounded by $\min_{th}$ and $\max_{th}$.

5.4. $W_c^* < Conn$ (Small pipe case)

Figure 21 plots the TCP behavior under RED queue management in the small pipe case, which is radically different from the large pipe case. There is still no apparent global or local synchronization. However, the bandwidth sharing becomes very unfair. Some flows completely get shut off, but the number of shut-off connections is smaller compared to the case with dropping tail gateways. Moreover, the buffer occupancy oscillates rapidly between 0 and around 2*$\max_{th}$, the latter of which is the threshold used by RED with the *gentle_* modification [19].

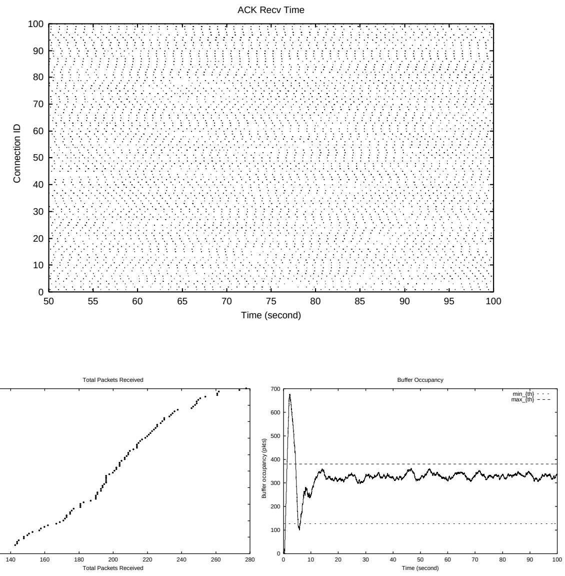


Figure 20. TCP behavior under RED queue management, in the large pipe case: 100 concurrent connections compete for T1 link with one-way propagation delay of 50 ms and buffer size of 760 packets. The RED configuration is RED-1 ($W_c^* = W_{opt} + \max_{th}$=420 packets).

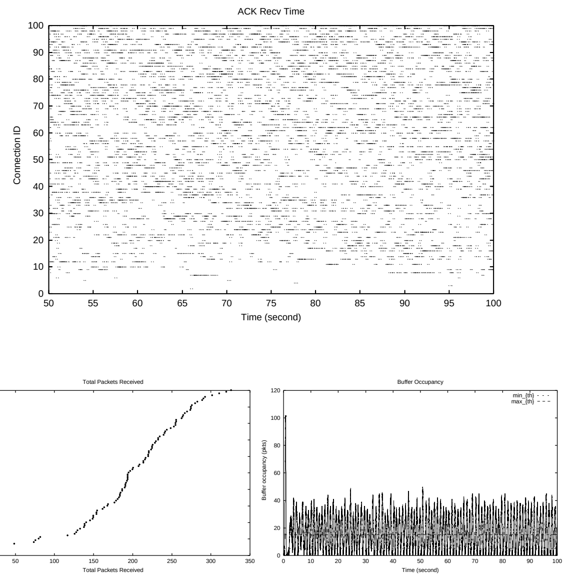


Figure 21. TCP behavior under RED queue management, in the medium pipe case: 100 concurrent connections compete for T1 link with one-way propagation delay of 50 ms and buffer size of 560 packets. The RED configuration is RED-3 ($W_c^* = W_{opt} + \max_{th}$= 55 packets).

5.5. $Conn < W_c^* < 3 * Conn$(Medium pipe case)

Figure 22 illustrates the TCP behavior under RED queue management in the medium pipe case, which is somewhat in between. Again, there is no obvious global or local synchronization. The bandwidth sharing is reasonably fair, with the number of packets received by different flows ranging from 50 to 350. The queue length oscillates around $\max_{th}$ periodically.

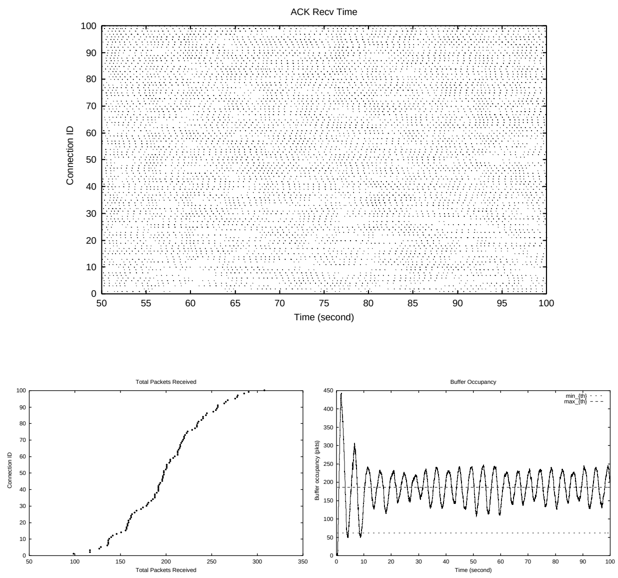
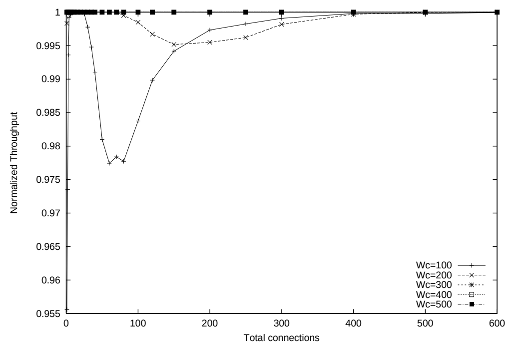


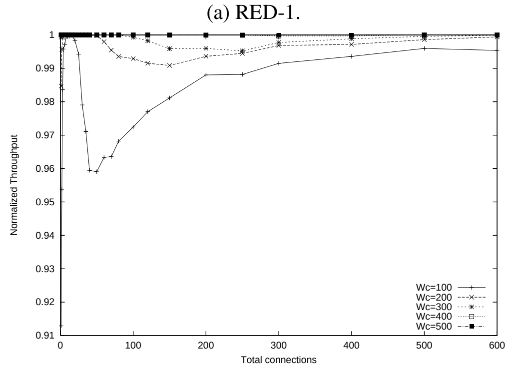
Figure 22. TCP behavior under RED queue management, in the medium pipe case: 100 concurrent connections compete for T1 link with one-way propagation delay of 50 ms and buffer size of 560 packets. The RED configuration is RED-2 ($W_c^* = W_{opt} + \max_{th}$= 223 packets).

5.6. Aggregate Throughput, Goodput & Loss Probability

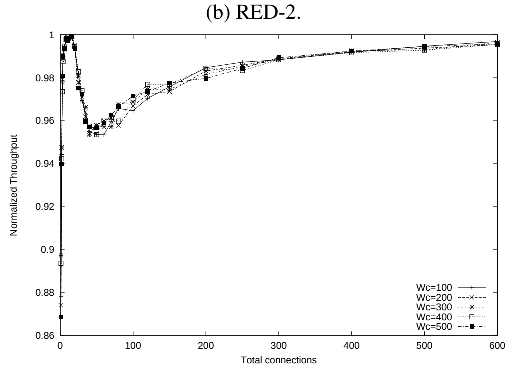
Figure 23 plots the aggregate throughput as a function of the number of competing TCP connections. As we can see, the throughput initially increases with the number of competing connections. As we further increase the number of TCP connections, the throughput decreases until a certain point beyond which the throughput keeps increasing again with the number of connections. A possible explanation for the non-monotonic relationship between the throughput and the number of connections is as follows. When the number of competing connections is small, the bottleneck link is not fully utilized. In this case, increase in the number of connections leads to more connections utilizing the link and higher throughput. Further increase in the number of connections results in more losses and timeouts. Consequently, most connections stay in slow start phase, which makes the bottleneck link under-utilized. However beyond a certain point, there are enough competing connections to keep the bottleneck link close to full utilization again, even though most connections stay in the slow start phase.



(a) RED-1.



(b) RED-2.

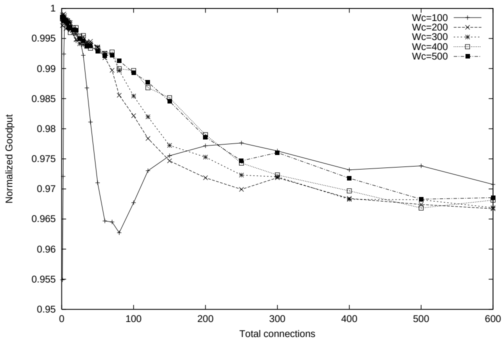


(c) RED-3.

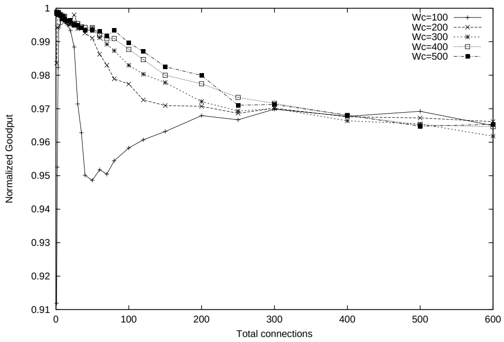
Figure 23. Throughput: varying number of connections compete for T1 bottleneck link with RED.

Figure 24 shows the aggregate goodput versus the number of connections. As with Drop-Tail, the increase in the number of competing connections leads to heavier congestion, and thereby more unnecessary retransmissions. As a consequence, the goodput decreases.

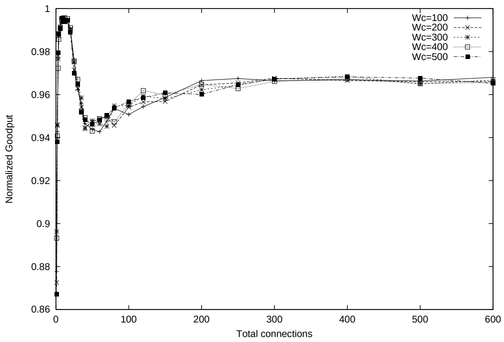
Figure 25 and Figure 26 plot the loss probability versus the number of connections, for a small number and a larger number of connections, respectively. As we can see, the loss rate increases quadratically with $Conn$ for small $Conn$, which conforms to the results reported in [32]. When the $Conn$ is large, the loss rate increases hyperbolically with $Conn$, just as in the case of Drop-Tail.



(a) RED-1.

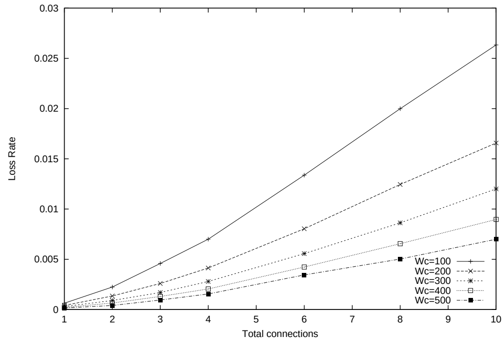


(b) RED-2.

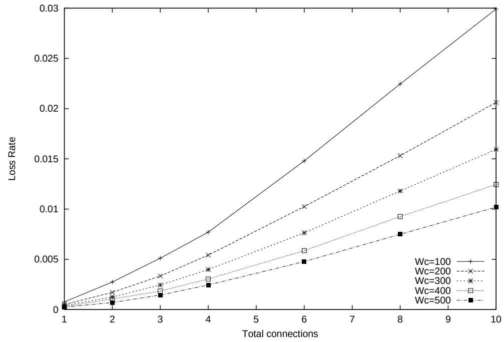


(c) RED-3.

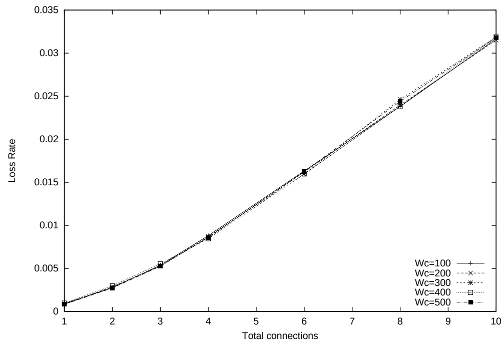
Figure 24. Goodput: varying number of connections compete for T1 bottleneck link with RED.



(a) RED-1.



(b) RED-2.



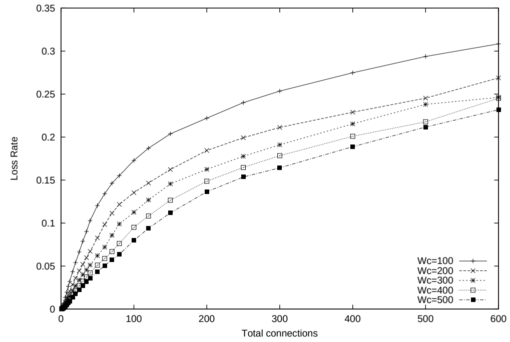
(c) RED-3.

Figure 25. Loss probability for a small number of connections competing for T1 bottleneck link with RED.

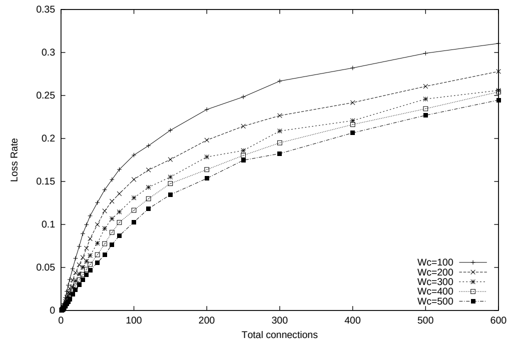
6. TCP/SACK Performance under Drop-Tail and RED

In this section, we study the TCP/Sack (TCP with support for *selective acknowledgments* (SACK)) under both Drop-Tail and RED dropping policies.

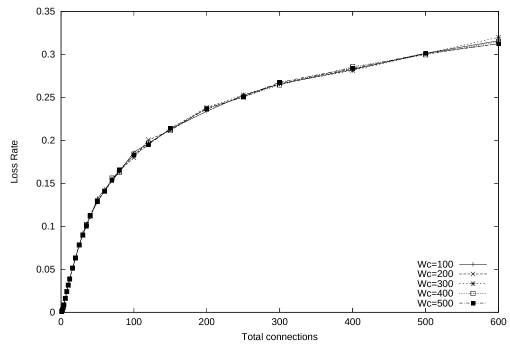
Figure 27 plots the aggregate throughput as a function of



(a) RED-1.



(b) RED-2.



(c) RED-3.

Figure 26. Loss probability for a large number of connections competing for T1 bottleneck link with RED.

the number of competing TCP connections. The results are similar to TCP/Reno for possibly the same reason.

Figure 28 shows the aggregate goodput versus the number of TCP/SACK connections. As in the case of TCP/Reno, the aggregate goodput decreases with the number of connections due to the increase in the total number of unnecessary retransmissions. On the other hand, the goodput of TCP/SACK is slightly higher than that of TCP/Reno due to more efficient loss recovery mechanism in TCP/SACK.

Figure 29 and Figure 30 plot the aggregate loss probability of TCP/SACK for a small number and a large number of connections, respectively. As in TCP/Reno, the loss rate is proportional to $Conn^2$ for small $Conn$. For large $Conn$, the loss rate increases hyperbolically with $Conn$.

7. TCP behavior with different RTT

In this section, we explore the TCP performance under different RTT's. We start with a simple scenario in which there are only two different RTT values and all flows are long-lived FTP sessions that can reach the steady state (in Section 7.1). We then consider more complicated scenarios where there are more than two different RTT values (in Section 7.2).

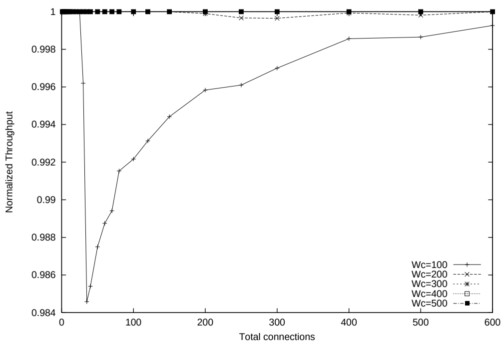
7.1 Bulk Transfers with 2 Different RTT's

We divide all TCP flows into two equal-sized groups G_1 and G_2. All flows within the same group G_i have the same two-way propagation delay PD_i (i = 1, 2). We vary PD_1 while keeping PD_2 fixed. This can be achieved by properly adjusting the one-way propagation delay on the high-bandwidth (10 Mbps) links in Figure 1. Figure 31 shows the network topology used in our simulations.

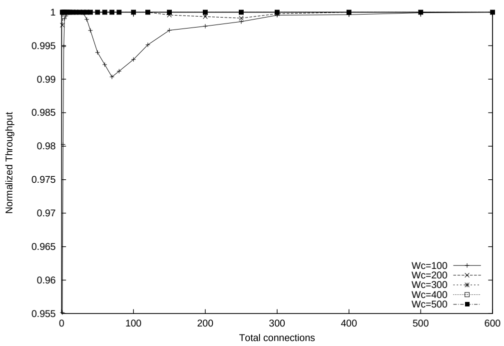
For each simulation run, we record the average queuing delay (Q) experienced by all packets. Then we can estimate the average roundtrip time for flows in group G_i, as $RTT_i = PD_i + Q$. We also record $Goodput_i$, the average goodput for flows in group G_i. The goal is to study how $(Goodput_2/Goodput_1)$ changes with respect to (RTT_1/RTT_2).

As suggested by Floyd *et al.* [15], in order to explore properties of network behavior unmasked by the specific details of traffic phase effects, *we always add a random processing time at the TCP sender*, which is uniformly chosen between zero and the bottleneck service time for a TCP data packet.

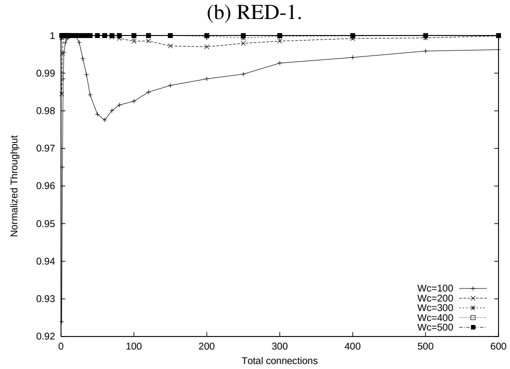
Fig. 32 shows the results of 6 competing FTP sessions with the Drop-Tail FIFO queue management. The bottleneck link has T1 capacity and 50 msec one-way propagation delay. PD_2 is kept at 102 msec. As we can see, the goodput ratio satisfies Equation (1) over a wide range of RTT



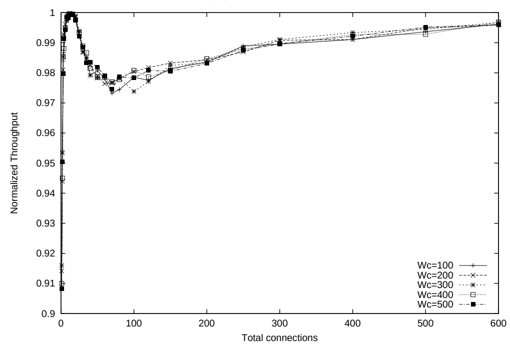
(a) Drop Tail.



(b) RED-1.

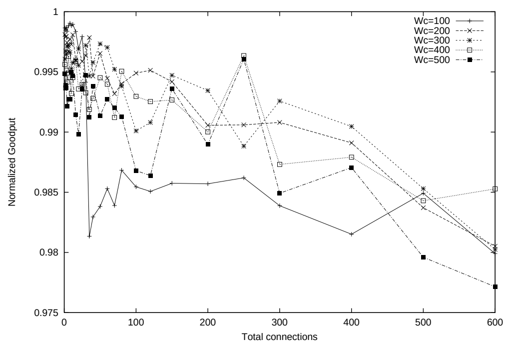


(c) RED-2.

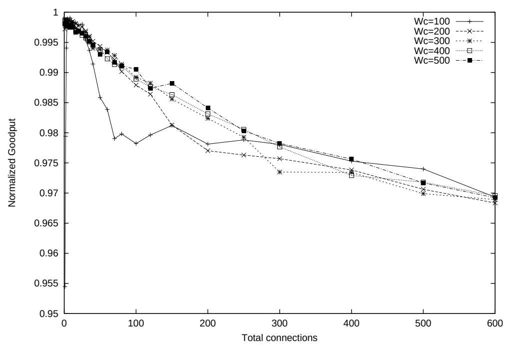


(d) RED-3.

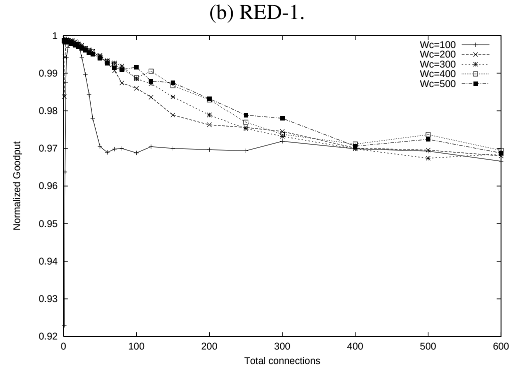
Figure 27. Throughput: varying number of TCP/SACK connections compete for T1 bottleneck link.



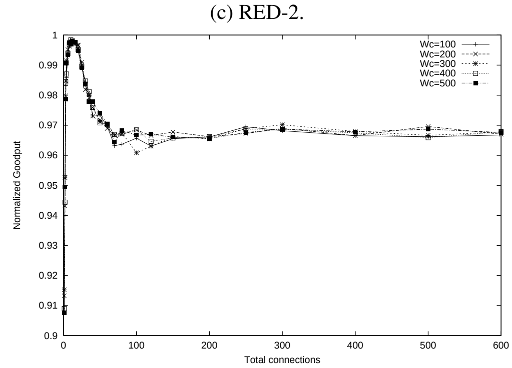
(a) Drop Tail.



(b) RED-1.

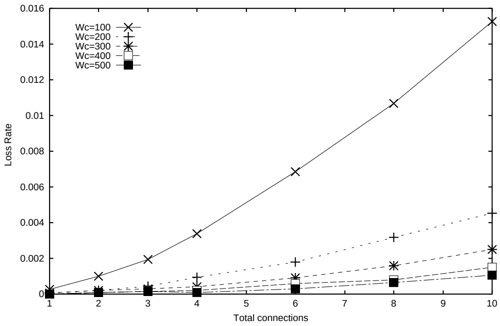


(c) RED-2.

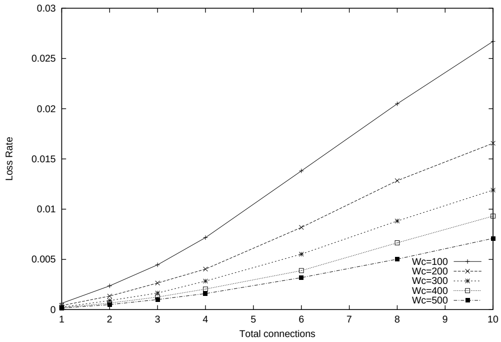


(d) RED-3.

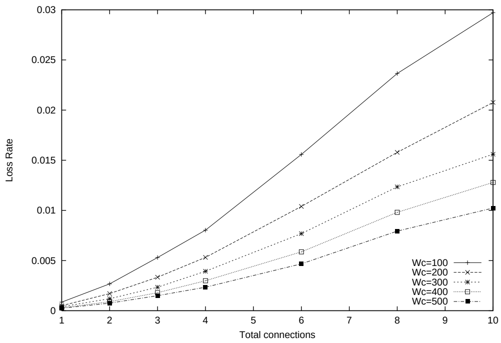
Figure 28. Goodput: varying number of TCP/SACK connections compete for T1 bottleneck link.



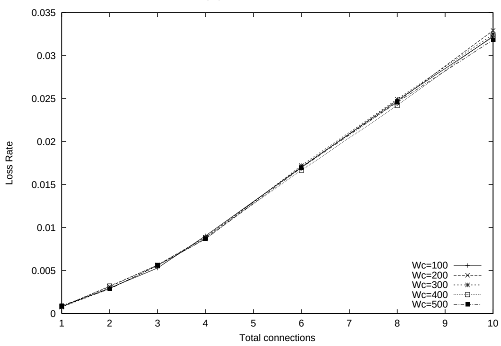
(a) Drop Tail.



(b) RED-1.

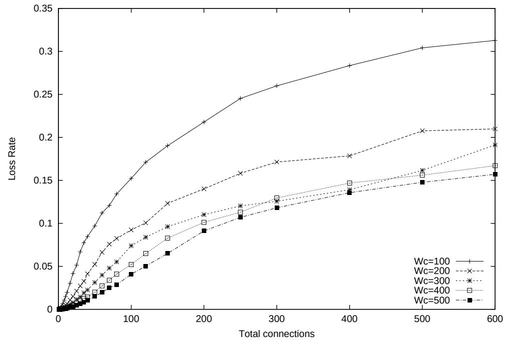


(c) RED-2.

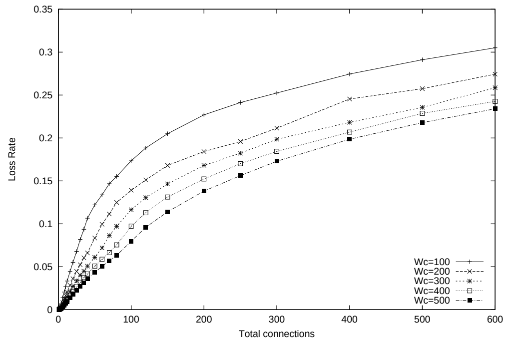


(d) RED-3.

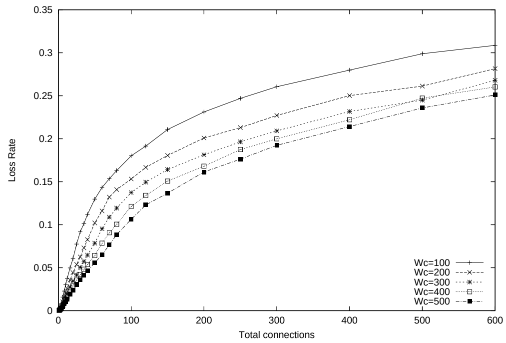
Figure 29. Loss probability for a small number of TCP/SACK connections competing for T1 bottleneck link.



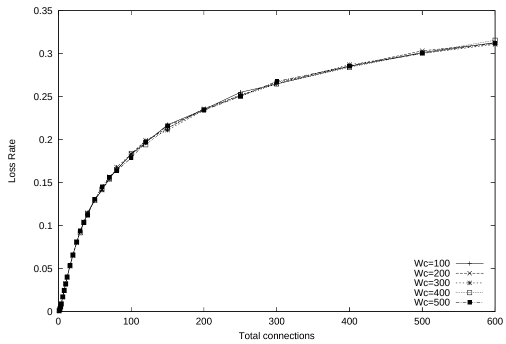
(a) Drop Tail.



(b) RED-1.



(c) RED-2.



(d) RED-3.

Figure 30. Loss probability for a large number of TCP/SACK connections competing for T1 bottleneck link.

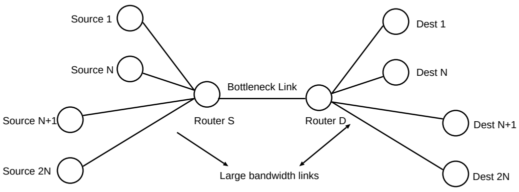


Figure 31. Network topology with 2 different RTT's.

ratio, in particular, when RTT ratio is less than 5. (Extensive simulations with many different settings also give the same result.)

$$0.5 \times \left(\frac{RTT1}{RTT2}\right)^2 \leq \frac{Goodput_2}{Goodput_1} \leq \left(\frac{RTT1}{RTT2}\right)^2 \quad (1)$$

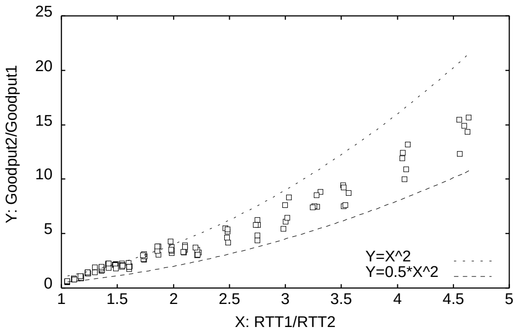


Figure 32. The effect of Drop-Tail queue (dt in Table 3) on 6 FTP sessions with 2 different RTT's. The bottleneck link has T1 capacity and 50 msec one-way propagation delay. PD_1 varies while PD_2 is kept at 102 msec.

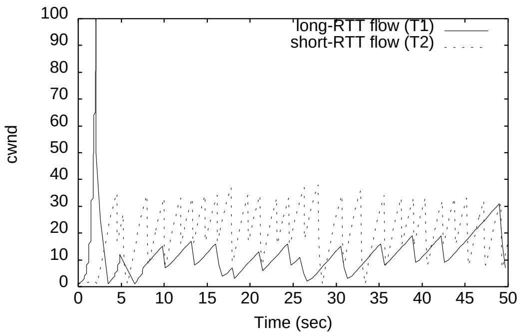


Figure 33. The congestion window evolution of two TCP flows T_1 and T_2 with Drop-Tail FIFO queue management. $RTT_1 > RTT_2$.

To explain (1), we need to thoroughly understand the synchronization effect of Drop-Tail gateways. For brevity, let us only consider two competing TCP flows. All our analysis still applies when there are more than two competing TCP flows. Fig. 33 illustrates the evolution of the congestion window size ($cwnd$) for two competing TCP flows T_1 and T_2 (with $RTT_1 > RTT_2$) under Drop-Tail FIFO queue management. As we can see, the synchronization effect on T_1 and T_2 is more complicated than the global synchronization for TCP connections with the same RTT. For T_2, whenever the bottleneck buffer becomes full, it will get a loss from 0 up to one roundtrip time (RTT_2) after it increases its congestion window. It takes another roundtrip for T_2 to detect the loss by 3 duplicated ACK's and reduce its sending rate by halving its $cwnd$. It takes one more roundtrip for such rate reduction to actually take effect. Therefore, on average it takes roughly $2.5 \times RTT_2$ for the rate of T_2's packets arriving at the bottleneck router to decrease. When T_1 increases its $cwnd$ and sends one more packet, this extra packet won't get dropped unless it arrives at the bottleneck queue before the packet arrival rate of T_2 decreases. With the random processing time at the sender, this extra packet can arrive at the bottleneck queue at any time within RTT_1 after the queue becomes full. Therefore, we can estimate the drop probability for this extra packet as $\min(1, 2.5 \times RTT_2/RTT_1)$, which is within $[0.5, 1]$ when $RTT_1/RTT_2 \in [1, 5]$. Or equivalently,

$$\frac{\text{\#drops seen by } T_1 \text{ in unit time}}{\text{\#drops seen by } T_2 \text{ in unit time}} \in [0.5, 1] \quad (2)$$

Assuming that at steady state, the $cwnd$ of connection T_i grows from W_i to $2 \times W_i$ during each epoch, then T_i sees a drop every $W_i \times RTT_i$. Consequently, (2) becomes

$$\frac{W_2 \times RTT_2}{W_1 \times RTT_1} \in [0.5, 1] \quad (3)$$

Meanwhile, the goodput for T_i can be approximated as:

$$Goodput_i = \frac{3 \times W_i}{2 \times RTT_i} \quad i = 1, 2. \quad (4)$$

From (3) and (4), we immediately get (1). Note that as we further increase RTT ratio (beyond 5), the gap between the actual goodput ratio and (1) enlarges. Part of the reason may be that the increasing effect of timeouts tends to break the synchronization.

Now we consider the effect of RED queue management. It is well-known that $Goodput_i$ is roughly inversely proportional to $RTT_i\sqrt{p_i}$, where p_i is the packet loss rate for flow i [35]. With RED, different flows roughly experience the same packet loss rate under steady state. Consequently, we have:

$$\frac{Goodput_2}{Goodput_1} \approx \frac{RTT_1}{RTT_2} \quad (5)$$

Our simulation results confirm (5), as shown in Figure 34.

7.2 Bulk Transfers with 8 Different RTT's

In this section, we consider 8 competing FTP sessions F_i (i=1,2,...8) all with different RTT's. The two-way propagation delay for F_1 is kept at 102 msec. The propagation

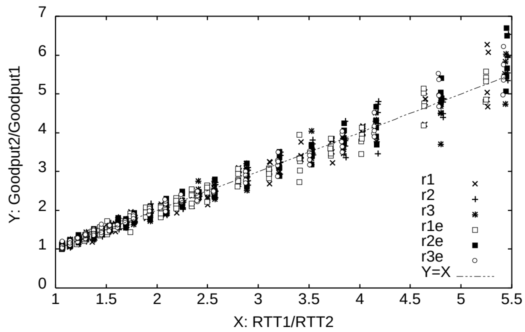


Figure 34. The effect of RED gateway on FTP sessions with 2 different RTT's. The simulation settings are the same as in Fig. 32 except that the bottleneck router uses RED queue management.

delays for all the other FTP sessions are uniformly chosen between 102 msec and 300 msec during each simulation run. The configurations for the bottleneck link and the buffer size are the same as before. For each run of the experiment, we record goodput ratios $Goodput_1/Goodput_i$ and the RTT ratios RTT_i/RTT_1 (i = 2,3,...,8). We then make a scatter plot of all the data points (RTT_i/RTT_1, $Goodput_1/Goodput_i$) obtained from 15 runs for each queue management scheme. The results are summarized in Fig. 35.

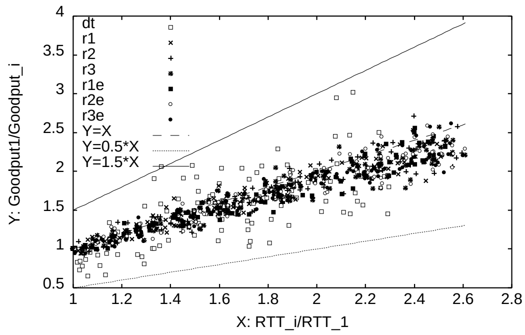


Figure 35. Comparison between the effect of RED gateway and the dropping tail gateway on 8 TCP flows with all different RTT's.

As we can see, with Drop-Tail, the bias against long-RTT flows with 8 different RTT's is smaller than in the case of 2 different RTT's. The throughput ratios are centered around line $Y = X$ and are well-bounded by two lines $Y = 1.5X$ and $Y = 0.5X$. In comparison, with RED, the throughput ratios are clustered much closer to line $Y = X$. This suggests that RED is more fair than Drop-Tail. To quantify the fairness of different queue management schemes for flows with different RTT's, we use the *normalized fairness ratio* [1], which is defined as follows.

$$f = \frac{(\sum_{i=1}^{n} Goodput_i \times RTT_i)^2}{n \sum_{i=1}^{n} (Goodput_i \times RTT_i)^2}$$

Fig. 36 shows the normalized fairness ratio for different queue management policy. As we can see that Drop-Tail is significantly less fair than RED.

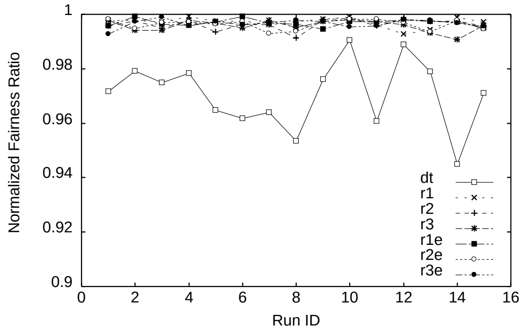


Figure 36. Normalized fairness ratio using different queue management policies.

To summarize, for bulk transfers RED tend to reduce the bais against long-RTT connections. With both RED and Drop-Tail queue management, a connection's goodput is inversely proportional to its RTT.

8. Related Work

Large scale performance analysis has been an active research area recently. Many researches focus on building scalable simulators, such as [3, 9, 21, 23, 25, 26, 6, 46, 47]. For example, [9] speeds up simulation by using parallel simulation techniques. It does not change the underlying simulation paradigm. Other approaches focus on raising the level of simulation abstraction. In particular, [3] proposes to abstract an entire group of closely spaced packets as a packet train. This enables their simulator to represent network traffic with considerably fewer events than a traditional simulator. Two other abstraction techniques: centralized computation and abstract packet distribution, are suggested in [23], which are shown to apply to multicast simulation. The combination of these two methods leads to orders of magnitude of performance improvement without sacrificing accuracy.

Another approach is to use fluid models to approximate real traffic at coarser time scales. For instance, [25, 26] track events corresponding to the changes in the fluid rate. [46, 47] apply the fluid model through time-driven fluid simulation (TDFS). TDFS discretizes time into small time intervals, and assumes a constant fluid rate during each time interval. The system states are updated periodically. Simulation speedup can be achieved by using a larger time interval. As pointed out in [21], the fluid model based approaches lack packet information, which makes it difficult to simulate packet based protocols like TCP. [21] addresses this issue by developing time-stepped hybrid simulation (TSHS). TSHS discretizes time into small time intervals. Packets from the same session falling into the same timestep are grouped into a chunk, and assumed to be evenly distributed. This technique relieves the simulator from keeping track of each individual packet.

Analyzing simulation results to estimate TCP performance, as done in this paper, is a very different approach from building a scalable simulator. The strategy taken by [32] is the closest to ours. It studies how TCP throughput, loss rates, and fairness are affected by changing the number of flows. Their work differs from ours in that we focus on studying how TCP performance changes by varying different parameters of the network model, whereas they focus on studying how the TCP behaves as one of the parameters - the number of flows changes, while fixing all the other parameters. Furthermore they study TCP/Tahoe under RED gateways, whereas we study TCP/Reno under either drop-tail or RED gateways. RED dropping policy is not sensitive to instantaneous queue occupancy, so it is relatively easy to obtain the steady state performance.

There has been a vast volume of research on studying RED dropping policy, ranging from analytical modeling [5, 31] to simulations and experimental evaluation [45, 10, 8, 48], from recommendations for parameters and architecture configuration [18, 19, 13, 22] to proposals of alternatives, which include BLUE [12], SRED (Stabilized RED) [34], Adaptive RED [11], FRED (Fair Random Early Drop) [28], and BRED (Balanced RED) [4]. Our simulation study of RED performance complements the existing body of research by comparing the aggregate throughput, goodput, loss rate, and roundtrip bias of TCP connections under RED gateways with those under drop-tail gateways. Our results show that for long-lived flows, the aggregate throughput, goodput, and loss rate under drop-tail gateways are comparable to those under RED gateways. On the other hand, RED tend to reduce the bias against long-RTT connections, making the bandwidth sharing more fair than drop-tail.

There are lots of other studies on TCP performance. Earlier ones include [14, 27, 30] etc. Recently, there have been significant research efforts on developing analytical models for the dynamic behavior of TCP. [35] and [36] develop analytical models for predicting the steady state sending rate of a bulk transfer TCP flow as a function of loss rate and round trip time. [7] extends the steady state model in [35] by considering the startup effects. Their extended model characterizes the expected value and distribution of TCP connection establishment and data transfer latency as a function of transfer size, round trip time, and packet loss rate. [2] develops a stochastic model for the performance of a single TCP session in the presence of random packet loss. [20] presents a closed queueing network model for the estimation of the performance of TCP/Tahoe connections, such as packet loss rate, $cwnd$, and $ssthresh$. We believe our simulation studies are complementary to the analytical modeling, and help further explore TCP dynamics under controlled environment.

9. Conclusion and Future Work

In this paper, we use extensive simulations to systematically investigate the performance of many TCP flows. Our major findings are:

- When all the connections have the same propagation delay, without adding random processing time, global synchronization occurs when $W_c > 3 * Conn$. When $W_c < Conn$, the bandwidth sharing becomes extremely unfair: some get most bandwidth, while others become completely shut-off. When the value of $\frac{W_c}{Conn}$ is in between, the performance falls in between: there is local synchronization, but no global synchronization nor shut-off connections.
- Adding random processing time or using RED gateways makes both synchronization and consistent discrimination less pronounced.
- For both Drop-Tail and RED gateways, the aggregate throughput of TCP flows is close to link bandwidth, except when there are too few connections to fill up the pipe. There is a small variation in the throughput as the number of connections increases, but most variation is within 2%.
- For both Drop-Tail and RED gateways, the aggregate goodput of TCP flow decreases slightly with the number of connections due to the increase in the unnecessary retransmissions. The only exception occurs when there are too few connections to fill up the pipe.
- For both Drop-Tail and RED gateways, the aggregate loss rate of TCP flow is proportional to $Conn^2$ for small $Conn$, and increases hyperbolically with $Conn$ for large $Conn$.
- TCP/SACK helps to increase the aggregate goodput slightly compared to TCP/Reno, due to its more efficient loss recovery mechanism.

Based on our results, we make four observations. First, since adding random processing time makes both synchronization and consistent discrimination less pronounced, these dramatic behaviors are not severe problems in the real Internet. Second, for bulk transfers, the aggregate throughput, goodput, and loss rate under Drop-Tail gateways are comparable to those under RED gateways. Third, TCP/SACK helps to slightly increase the goodput compared to TCP/Reno. On the other hand, SACK cannot completely avoid unnecessary retransmissions. For example, when the timeout value is smaller than it should be, a packet may be re-sent even though it has arrived at the receiver. Finally, we observe that for bulk transfers, RED tends to reduce the bias against long-RTT connections, making the bandwidth sharing more fair than with Drop-Tail.

There are a number of directions for future work. First, we have shown the loss probability curves can be approximated quite well with simple analytical functions. As part of our future work, we will investigate how to quantitatively determine the parameters in the functions. Second, we plan to further explore TCP performance under multiple-bottleneck networks. Third, in this paper, we mainly focus on studying the performance of many long-lived TCP flows. We are very interested in investigating the performance of HTTP flows in the future. Finally, we plan to use Internet experiments to verify some of the results in the paper.

References

[1] A. Aagarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. *Proc. IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

[2] A. A. Abouzeid, S. Roy, and M. Azizoglu. Stochastic Modeling of TCP over Lossy Links. *Proc. of IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

[3] J. Ahn and P. B. Danzig. Packet network simulation: speedup and accuracy versus timing granularity. IEEE/ACM Transactions on Networking, Volume 4, No. 5, pp. 743-757, October 1996.

[4] F. Anjum and L. Tassiulas. Balanced-RED: An Algorithm to Achieve Fairness in the Internet. *Proc. IEEE INFOCOM'99*, New York City, USA, March 1999.

[5] T. Bonald, M. May, J. C. Bolot. Analytic Evaluation of RED Performance. *Proc. IEEE INFOCOM'2000*, Tel-Avi, Israel, March 2000.

[6] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in Network Simulation. IEEE Computer, 33(5), pp. 59-67, May 2000.

[7] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. *Proc. of IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

[8] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for Web Traffic. *Proc. ACM SIGCOMM'2000*, Stockholm, Sweden, August 2000.

[9] J. Cowie, D. Nicol, and A. Ogielski. Modeling the Global Internet. Computing in Science & Engineering, pp. 30-38, 1999.

[10] O. Elloumi and H. Afifi. RED Algorithm in ATM Networks. Technical Report, June 1997.

[11] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. *Proc. IEEE INFOCOM'99*, New York City, USA, March 1999.

[12] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A New Class of Active Queue Management Algorithms. University of Michigan Technical Report CSE-TR-387-99, April 1999.

[13] V. Firoiu, and M. Borden. A Study of Active Queue Management for Congestion Control. *Proc. IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

[14] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. Computer Communication Review, Vol.21, No.5, October 1991, p. 30-47.

[15] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. Internetworking: Research and Experience, V.3 N.3, September 1992, p.115-156.

[16] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.

[17] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10-23, October 1994.

[18] S. Floyd. RED: Discussions of Setting Parameters. http://www.aciri.org/floyd/REDparameters.txt , November 1997.

[19] S. Floyd. Recommendation on Using the "gentle_" Variant of RED. http://www.aciri.org/floyd/red/gentle.html, March 2000.

[20] M. Garetto, R. Lo Cigno, M. Meo, M. Ajmone Marsan. A Detailed and Accurate Closed Queueing Network Model of Many Interacting TCP Flows. *Proc. IEEE INFOCOM'2001*, Anchorage, AK, USA, April 2001.

[21] Y. Guo, W. Gong, and D. Towsley. Timing-stepped Hybrid Simulation (TSHS) for Large Scale Networks. *Proc. IEEE INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

[22] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. A Control Theoretic Analysis of RED. *Proc. IEEE INFOCOM'2001*, Anchorage, AK, USA, April 2001.

[23] P. Huang, D. Estrin, and J. Heidemann. Enabling Large-scale Simulations: Selective Abstraction Approach to the Study of Multicast Protocols. *Proc. Sixth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, Montreal, Canada, July 1998.

[24] V. Jacobson and M. Karels. Congestion Avoidance and Control. *Proc. ACM SIGCOMM '88*, Stanford, CA, USA, August 1988.

[25] G. Kesidis, A. Singh, D. Cheung, and W. W. Kwok. Feasibility of Fluid-Driven Simulation for ATM Network. *Proc. IEEE GLOBECOM'96*, pp. 2013-2017, Vol. 3, Westminster, London, Britian, November 1996.

[26] K. Kumaran and D. Mitra. Performance and Fluid Simulations of a Novel Shared Buffer Management System. *Proc. IEEE INFOCOM'98*, San Francisco, CA, USA, March 1998.

[27] T. V. Lakshman and U. Madhow. Performance Analysis of Window-based Flow Control using TCP/IP: Effect of High Bandwidth-Delay Products and Random Loss. *Proc. IFIP TC6/WG6.4 Fifth International Conference on High Performance Networking*, June 1994.

[28] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proc. ACM SIGCOMM'97*, Cannes, France, September 1997.

[29] B. Mah. An Empirical Model of HTTP Network Traffic. *Proc. IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[30] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, April 1996.

[31] V. Misra, W. Gong, and D. Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *Proc. ACM SIGCOMM'2000*, Stockholm, Sweden, August 2000.

[32] R. Morris. TCP Behavior with Many Flows. *Proc. IEEE International Conference on Network Protocols'97*, Atlanta, GA, USA, October 1997.

[33] UCB/LBNLVINT Network Simulator - `ns` (version 2). http://www-mash.cs.berkeley.edu/ns, 1997.

[34] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proc. IEEE INFOCOM'99*, New York City, USA, March 1999.

[35] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proc. of ACM SIGCOMM'98*, Vancouver, BC, Canada, September 1998.

[36] J. Padhye, V. Firoiu, and D. Towsley. A stochastic Model of TCP Reno Congestion Avoidance and Control. CMPSCI Technical Report 99-02.

[37] V. N. Padmanabhan and R. H. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. In *Proc. of IEEE Globecom'98*, Sydney, Australia, November 1998.

[38] L. Qiu, Y. Zhang, S. Keshav. On individual and aggregate TCP Performance. In *Proc. of ICNP'99*, Toronto, Canada, November 1999.

[39] V. Rosolen, O. Bonaventure and G. Leduc. Impact of Cell Discard Strategies on TCP/IP in ATM UBR Networks. *Proc. 6th Workshop on Performance Modelling and Evaluation of ATM Networks (IFIP ATM '98)*, July 1998.

[40] V. Rosolen, O. Bonaventure and G. Leduc. A RED Discard Strategy for ATM Networks and Its Performance Evaluation with TCP/IP Traffic. *ACM Computer Communication Review*, July 1999.

[41] L. Qiu, Y. Zhang, and S. Keshav. On Individual and Aggregate TCP Performance. Cornell CS Technical Report, TR99-1744, May 1999.

[42] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, January 1999.

[43] K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transaction on Computer Systems*, 8(2):158-181, May 1990.

[44] S. Shenker, L. Zhang, and D. D. Clark. Some Observations on the Dynamics of a Congestion Control Algorithm. ACM Computer Communication Review, pp.30-39, 1990.

[45] C. Villamizar and C. Song. High Performance TCP in ANSNET. *ACM Computer Communications Review*, 24(5):45-60, October 1994.

[46] A. Yan, and W. B. Gong. Time-Driven Fluid Simulation for High-Speed Networks With Flow-Based Routing. *Proc. of the Applied Telecommunications Symposium'98*, pp.153-158, Boston, MA, USA, Aprial 1998.

[47] A. Yan, and W. B. Gong. Fluid Simulation for High Speed Networks. IEEE Transactions on Information Theory, June 1999.

[48] Y. Zhang and L. Qiu. Understanding the End-to-End Performance Impact of RED in a Heterogeneous Environment. Cornell CS Technical Report 2000-1802, July 2000.