Techniques for Eliminating Packet Loss in Congested TCP/IP Networks

Wu-chang Feng† Dilip D. Kandlur‡ Debanjan Saha‡ Kang G. Shin†

†Department of EECS
University of Michigan
Ann Arbor, MI 63130
Phone: (313) 763-5363 Fax: (313) 763-4617
{*wuchang,kgshin*}*@eecs.umich.edu*

‡Network Systems Department
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
Phone: (914) 784-7194 Fax: (914) 784-6205
{*kandlur,debanjan*}*@watson.ibm.com*

Abstract

The congestion control mechanisms used in TCP have been the focus of numerous studies and have undergone a number of enhancements. However, even with these enhancements, TCP connections still experience alarmingly high loss rates, especially during times of congestion. To alleviate this problem, the IETF is considering active queue management mechanisms, such as RED, for deployment in the network. In this paper, we first show that even with RED, loss rates can only be reduced marginally in times of congestion. We then show that the solution to this problem lies in decoupling congestion notification from packet loss through the use of explicit congestion notification (ECN) and in understanding the traffic generated by an aggregation of a large number of sources. Given this, we then propose and experiment with more adaptive active queue management algorithms (Adaptive RED) and more conservative end-host mechanisms (SUBTCP) which can significantly reduce loss rates across congested links. When used together, these mechanisms can effectively eliminate packet loss while maintaining high link utilizations under the most difficult scenarios.

1 Introduction

It is important to avoid high packet-loss rates in the Internet. When a packet is dropped before it reaches its destination, all of the resources it has consumed in transit have been wasted. In extreme cases, this situation can lead to congestion collapse [12]. Over the last decade, TCP and its congestion control mechanisms have been instrumental in controlling packet loss and in preventing congestion collapse across the Internet. Optimizing the congestion control mechanisms used in TCP has been one of the most active areas of research in the past few years. While a number of proposed TCP enhancements have made their way into actual implementations, TCP connections still experience high loss rates. Loss rates are especially high during times of heavy congestion, when a large number of connections compete for scarce network bandwidth. Recent measurements have shown that the growing demand for network bandwidth has driven loss rates up across various links in the Internet [21].

One of the reasons for high packet-loss rate is the failure of the network to provide early congestion notification to sources. This has led to proposals for active queue management, such as Random Early Detection (RED) [1, 11] and variations thereof [16]. While RED certainly outperforms traditional drop-tail queues, our experiments show that it is difficult to parameterize RED queues to perform well under different congestion scenarios. The key problem is that congestion notification does not directly depend on the number of connections multiplexed across the link. In order for early detection to work, congestion notification must be given at a rate sufficient to prevent packet loss due to buffer overflow, but not too high to cause underutilization of the bottleneck link. Section 3 examines this problem and shows how RED queues can self-parameterize themselves depending on traffic load in order to reduce packet loss and maintain high link utilization.

While adaptive queue-management techniques can provide some benefit, high loss rates at heavy loads are, in part, an artifact of TCP's congestion control algorithm. In steady state, TCP uses a window-based algorithm which multiplicatively decreases its transmission rate in response to congestion and linearly increases it otherwise. Unfortunately, when the window sizes are small, which is typically the case at times of congestion, a linear increase in window size has non-linear impact on a connection's transmission rate. When a large number of small TCP connections share a common bottleneck, the traffic generated can cause rapid fluctuations in queue lengths which result in packet loss. Section 4 investigates several ways of modifying TCP's congestion control mechanism in order to make the aggregate traffic generated by a large number of TCP connections better-behaved. In particular, using a scaled linear increase or a bandwidth-based linear increase in the window size, instead of TCP's current linear increase used by TCP, is shown to substantially reduce packet losses. When used together, the adaptive queue-management mechanisms and the proposed enhancements to TCP's windowing algorithm can effectively eliminate packet loss even in highly-congested networks. Section 5 presents extensive simulation results to demonstrate this. Finally, Section 6 concludes the paper with a discussion of possible extensions.

2 Background

When a network is congested, a large number of connections compete for a share of scarce link bandwidth. Over the last decade, TCP congestion control has been used to effectively regulate the rates of individual connections sharing network links. TCP congestion control is window-based. The sender keeps a congestion window whose size limits the number of unacknowledged packets the sender can have outstanding in the network. Upon receiving acknowledgments for successfully transmitted data, the sender increases its transmission rate by incrementing the size of its congestion window. At some point in time, the transmission rate eventually exceeds the network's capacity. When this happens, queues build up in the routers and overflow, causing packets to be dropped. TCP assumes that packet loss is due to congestion and reduces its congestion window upon detecting the loss.

The TCP congestion control algorithm is fairly straightforward. When a connection starts up, it attempts to ramp up its transmission rate quickly by exponentially increasing its congestion window. This stage is called *slow-start* and allows the source to double its congestion window, and thus its transmission rate, every round-trip time. In order to prevent excessive losses due to an exponentially-increasing transmission rate, TCP senders typically employ what is known as the congestion-avoidance algorithm [12]. In this algorithm, TCP keeps track of a threshold value (`ssthresh`) which is a conservative approximation of the window size which the network can support. When the window size exceeds this threshold, TCP enters the congestion avoidance phase. In this phase, the window is increased at a much slower rate of one segment per round-trip time. When the aggregate transmission rate of the active connections exceeds the network's capacity, queues build up and eventually packets are dropped. One way in which TCP detects a packet loss is through the receipt of a number of duplicate acknowledgments from the receiver [13]. Upon receiving a given number of duplicate acknowledgments, TCP infers that a packet loss has occurred and immediately halves its transmission rate by halving its congestion window. These mechanisms are called *fast retransmit* and *fast recovery*.

When congestion is severe enough such that packet loss cannot be inferred in such a manner, TCP relies on a retransmission timeout mechanism to trigger subsequent retransmissions of lost packets. When a retransmission timer is triggered, TCP reduces its window size to one segment and retransmits the lost segment. To prevent continual retransmissions in times of severe congestion and network outages, TCP employs an exponential back-off algorithm. In particular, if a TCP sender continually sends the same segment, but receives no acknowledgments for it, TCP doubles its retransmission timeout interval. Upon receipt of an acknowledgment for subsequent new segment, TCP resets the timeout interval and resumes its normal transmission.

One problem with the TCP congestion control algorithm over current networks is that the TCP sources back off their rates only after detecting packet loss due to queue overflow. This is a problem since considerable time may pass between the packet drop at the router and its detection at the source. In the meantime, a large number of packets may be dropped as the senders continue to transmission at a rate that the network cannot support. Active queue management has been proposed as a solution for preventing losses due to buffer overflow. The idea behind active queue management is to detect incipient congestion *early* and convey congestion notification to the end-hosts, allowing them to back off before queue overflow and packet loss occur. One form of active queue management being proposed by the IETF for deployment in the network is RED (Random Early Detection) [1,8]. RED maintains an exponentially-weighted moving average of the queue length which it uses to detect congestion. When the average queue length exceeds a minimum threshold (min_{th}), packets are randomly dropped or marked with an explicit congestion notification (ECN) bit [8, 22, 23] with a given probability. A connection receiving congestion notification in the form of an ECN mark, cuts its congestion window in half as it would if it had detected a packet loss. The probability that a packet arriving at the RED queue is either dropped or marked depends on, among other things, the average queue length, the time elapsed since the last packet was dropped, and an initial probability parameter (max_p). When the average queue length exceeds a maximum threshold (max_{th}), all packets are dropped or marked.

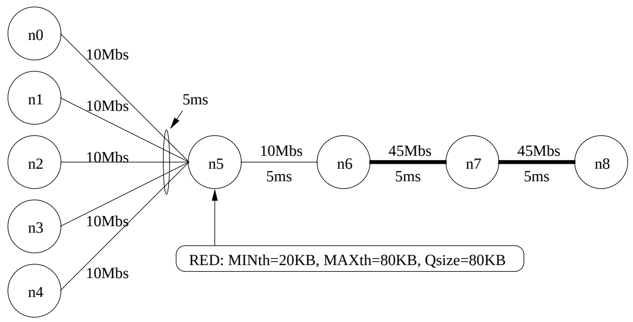


Figure 1: Network Topology

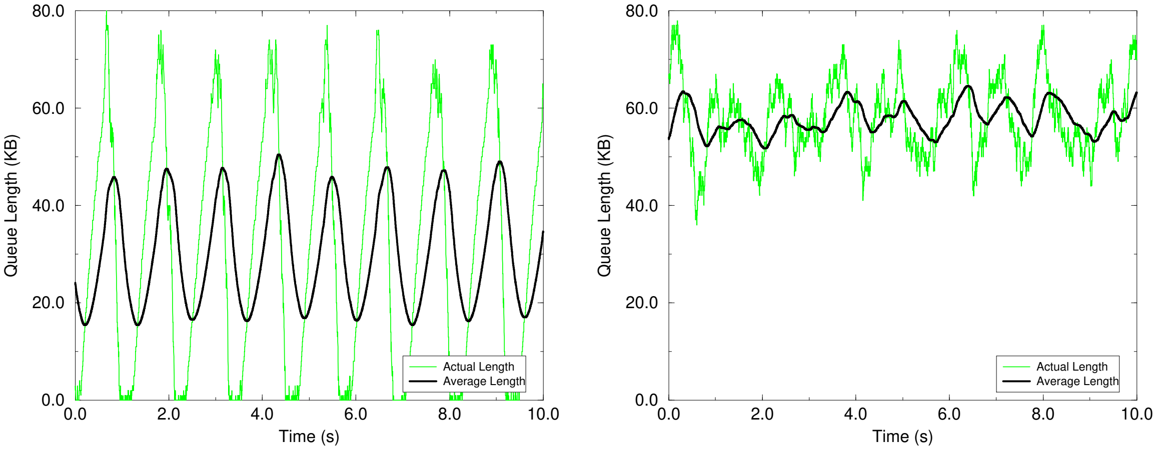
3 Active Queue Management

One of the inherent weaknesses of RED and some of the other proposed active queue-management schemes is that congestion notification does not directly depend on the number of connections multiplexed over the link. In order for early detection to work in heavily-congested networks, congestion notification must be given to enough sources so that the offered load is reduced sufficiently to avoid packet loss due to buffer overflow. Conversely, the RED queue must also prevent congestion notification from being given to too many sources in order to avoid situations where the bottleneck link becomes underutilized.

For example, consider a bottleneck link of capacity $10Mbs$ which is equally shared amongst several connections. Assuming TCP windowing, when 100 connections share the link, sending congestion notification to one connection reduces the offered load to $9.95Mbs$. On the other hand, when only 2 connections share the link, sending congestion notification to one of them reduces the offered load to $7.5Mbs$. In general, with a bottleneck link that supports N connections, giving congestion notification to one connection reduces the offered load by a factor of approximately $(1 - \frac{1}{2N})$. As N becomes large, the impact of individual congestion notifications decreases. Unless the RED algorithm is modified to be more aggressive, the RED queue degenerates into a simple drop-tail queue. On the other hand, as N becomes small, the impact of individual congestion notifications increases. In this case, unless the RED algorithm is modified to be less aggressive, underutilization can occur as too many sources back off their transmission rates in response to the observed congestion. This section examines the impact that traffic load has on active queue management techniques such as RED and proposes enhancements for optimizing performance.

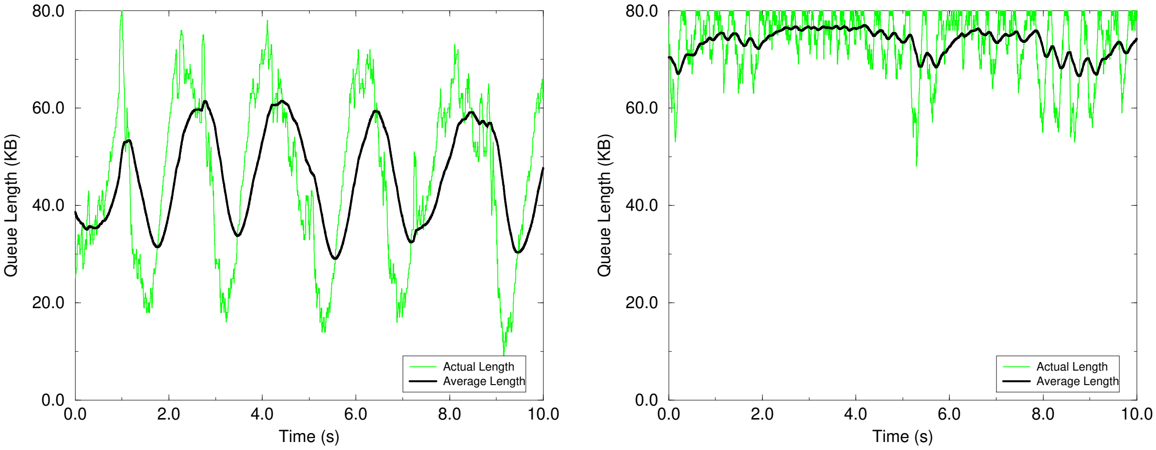
3.1 Traffic Load and Early Detection

To examine the impact that traffic load has on early detection mechanisms, we performed a set of experiments using the `ns` simulator [18]. We varied both the aggressiveness of the early detection algorithm and the total number of connections multiplexed on the bottleneck link. Figure 1 shows the network topology used in the experiments. Each connection originates at one of the leftmost nodes ($n0, n1, n2, n3, n4$) and terminates at $n8$, making the link between $n5$ and $n6$ the bottleneck. We first examine the performance of RED using ECN. By using RED and ECN enabled TCP sources, all packet losses from the RED queue can be attributed to buffer overflow. In order to isolate the effects of congestion notification triggered by min_{th} from that triggered by max_{th}, we set max_{th} to the queue size. This, in effect, disables max_{th} and causes packet loss to occur whenever early detection does not work. Additional experiments using max_{th} values which are below the queue



(a) 8 Connections (b) 32 Connections

Figure 2: Aggressive early detection ($max_p = 0.250$)



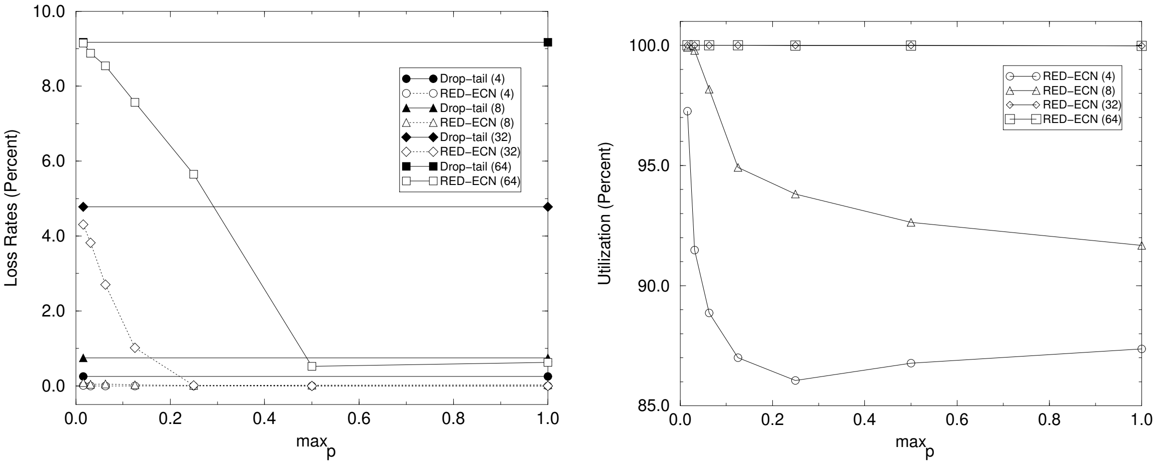
(a) 8 connections (b) 32 connections

Figure 3: Conservative early detection ($max_p = 0.016$)

size are described in Section 3.2.

Figure 2 shows the queue-length plot at the link between $n5$ to $n6$ when 8 and 32 connections are active. In these experiments, the RED algorithm is made aggressive by changing max_p, RED's initial drop probability. As Figure 2(a) shows, when only 8 connections are active, aggressive early detection sends congestion notification back to the sources at a rate which is too high, causing the offered load at times to be significantly smaller than the bottleneck link bandwidth. This causes periodic underutilization where the queue is empty and the bottleneck link has no packets to send. Figure 2(b) shows the queue-length plot when the number of connections is increased to 32. In this case, aggressive early detection performs as desired, sending congestion notification at a rate which can both avoid packet loss and achieve high link utilization.

Figure 3 shows the same set of experiments using a more conservative early detection. In contrast to Figure 2(a), Figure 3(a) shows that by using less aggressive early detection, the RED queue



(a) Packet loss rates (b) Bottleneck link utilization

Figure 4: Impact of early detection aggressiveness on RED-ECN

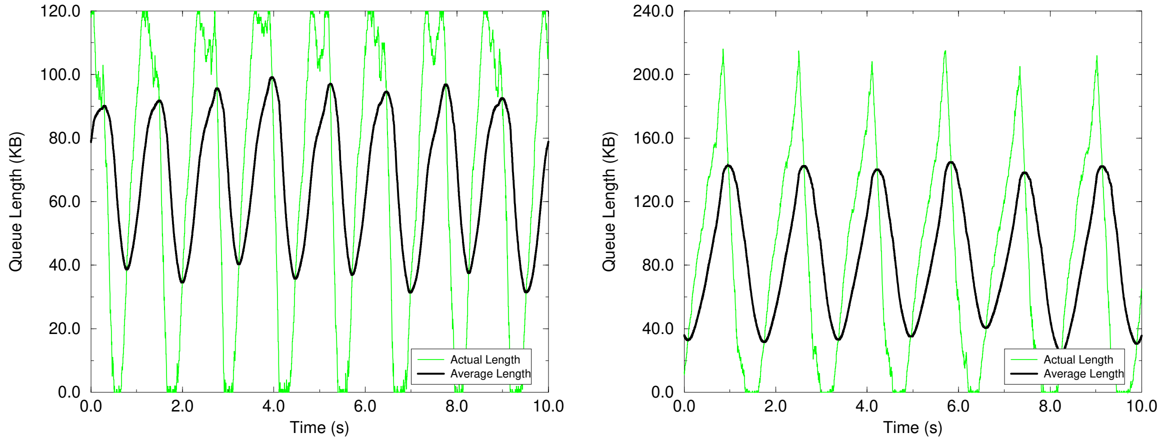
can maintain high link utilization while avoiding packet loss for smaller numbers of connections. However, when the number of connections is increased to 32, as Figure 3(b) shows, conservative early detection does not deliver congestion notification to sufficient number of sources. Thus, the RED queue continually overflows causing it to behave more like a drop-tail queue. The figure also shows that the bottleneck queue never drains even though it is dropping a significant number of packets. This indicates that the TCP sources do not back off sufficiently in response to congestion. In fact, the packets that are successfully delivered through the bottleneck queue trigger further increases in transmission rate. Consequently, the bottleneck queue remains close to fully occupied through the duration of the experiment.

To quantitatively evaluate the impact of max_p, we repeated the experiments across a range of traffic loads.[1] Figure 4(a) shows the loss rates observed for 4, 8, 32, and 64 connections. The figure also plots the loss rates when a drop-tail queue is used at the bottleneck link. As the drop-tail results show, loss rates at the bottleneck link increase proportionally to the number of connections using the link. The corresponding bottleneck link utilizations are shown in Figure 4(b). The figures show that for small numbers of connections, loss rates remain low across all values of max_p, while only small values of max_p can keep the bottleneck link at full utilization. Thus, to optimize performance over a small number of connections, early detection must be made conservative. In contrast, for large numbers of connections, bottleneck link utilizations remain high across all max_p values while only large values of max_p are able to prevent packet loss. In order to optimize performance in this case, early detection must be aggressive.

3.2 Avoiding Deterministic Congestion Notification

In the previous section, we set max_{th} equal to the queue size so that whenever the early detection algorithm fails, packet loss occurs. By setting max_{th} sufficiently below the queue size,

[1] In each experiment, connections are started within the first 10 seconds of simulation. After 100 seconds, both the loss rates and the link utilization for the bottleneck link are recorded for 100 seconds. The loss rate is calculated as the number of packets dropped by the queue divided by the total number of packets which arrive at the queue. The link utilization is calculated as the total number of packets sent divided by the maximum number of packets the link could send.



(a) queue size=120KB; max_{th}=80KB (b) queue size=240KB; max_{th}=80KB

Figure 5: Impact of max_{th} and queue size

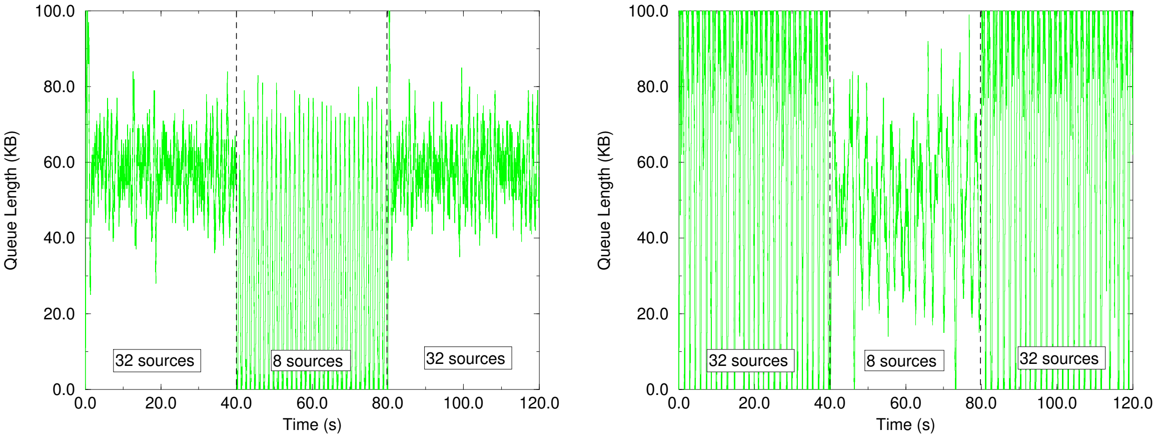
the RED algorithm can avoid packet losses when early detection fails by deterministically marking every incoming packet. Figure 5 shows the queue-length plot using the same experiment as in Figure 3(b) with a larger bottleneck queue size and a fixed max_{th} of 80KB. When the queue size is 120KB, the queue-length plot shows that even with a fairly significant amount of additional buffer space, packet loss is not eliminated. It also shows that the combined effect of using ECN and packet drops for signaling congestion notification leads to periods of time where TCP is able to impact the transmission rates of the sources. This is in contrast to the behavior seen in Figure 3(b). In that experiment, whenever a connection is able to send a packet through the bottleneck link it increases its transmission rate even though the bottleneck queue is full. By setting max_{th} sufficiently low and using ECN, all connections receive congestion notification when the queue is full whether it is from an ECN or from a packet loss. Thus, as Figure 5 shows, after a sustained period of ECN marking and packet loss, the sources back off enough to allow the queue to drain. One of the problems with deterministic marking is that it often goes overboard in signaling congestion to the end-hosts. As the queue-length plots in Figure 5 show, periods of congestion are immediately followed by fairly long periods of underutilization where the queue is empty. Furthermore, it takes quite a bit of extra buffer space in order to ensure that no loss occurs. Figure 5(b) shows the queue-length plot using a queue size of 240KB. As the figure shows, even when deterministic marking is done at average queue lengths of 80KB, the actual queue length can more than double before sources have a chance to back off.

3.3 Adaptive RED

From the discussion in the previous sections, it is clear that adapting RED parameters based on traffic load and mix can be beneficial to network performance. Figure 6 represents an on-line algorithm for adaptively changing the RED parameters according to the observed traffic. The idea behind this algorithm is to infer whether or not RED should become more or less aggressive by examining the variations in average queue length. If the average queue length oscillates around min_{th}, then the early detection mechanism is being too aggressive. If on the other hand, it oscillates around max_{th}, then the early detection mechanism is being too conservative. Based on the observed queue length dynamics, the algorithm adjusts value of max_p. In particular, it scales max_p by

Every $Q(ave)$ Update:
    if ($min_{th} < Q(ave) < max_{th}$)
        status = `Between`;
    if ($Q(ave) < min_{th}$ && status != `Below`)
        status = `Below`;
        $max_p = max_p \ / \ \alpha$;
    if ($Q(ave) > max_{th}$ && status != `Above`)
        status = `Above`;
        $max_p = max_p \ * \ \beta$;

Figure 6: Adaptive RED algorithm



(a) Aggressive detection ($max_p = 0.250$) (b) Conservative detection ($max_p = 0.016$)

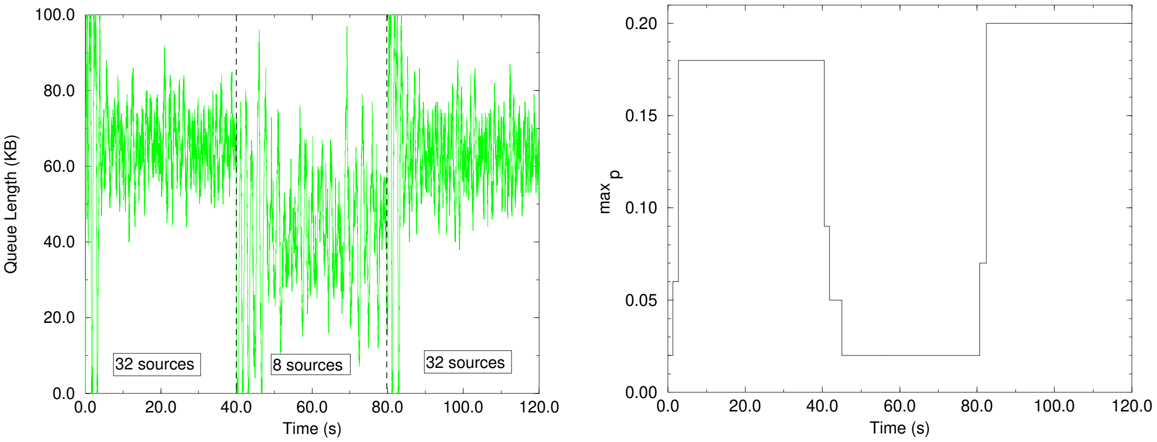
Figure 7: Static random early detection

constant factors of α and β depending on which threshold it crosses.

To explore the feasibility of the approach proposed in Figure 6, we ran another set of experiments using the same network as shown in Figure 1[2]. In this experiment, we vary the number of active connections between 8 and 32 over 40 second intervals. Figure 7 shows the queue-length plots using RED queues statically configured to be either aggressive or conservative. When aggressive early detection is used, as shown in Figure 7(a), the RED queue performs well whenever 32 connections are active. When only 8 connections are active, however, the RED queue is too aggressive in its congestion notification, thus causing periodic underutilization when the queue is empty. When conservative early detection is used, as shown in Figure 7(b), the RED queue only performs well when 8 connections are active. When all 32 connections are active, the RED queue continually fluctuates between periods of sustained packet loss and ECN marking and subsequent periods of underutilization.

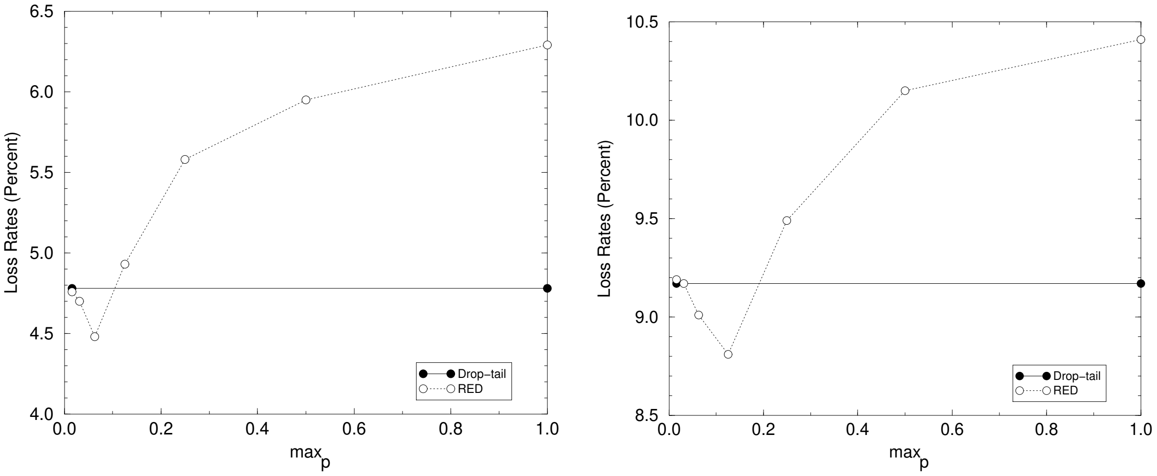
Figure 8(a) shows the queue-length plot of the same experiment using the adaptive RED algorithm

[2]Here the RED queues are of size $100KB$.



(a) Queue Length (b) max_p parameter

Figure 8: Adaptive random early detection



(a) 32 connections (b) 64 connections

Figure 9: Impact of early detection aggressiveness on RED

with α and β set to 3 and 2, respectively. We initially set max_p to 0.020 and then allow the algorithm to adjust the parameter accordingly. As the plot shows, after brief learning periods when the experiment starts and when the input traffic changes, the RED queue is able to adapt itself well. Figure 8(b) plots the max_p parameter as the RED queue adapts it to the input traffic. As expected, its value adapts to reflect the number of active flows. When all 32 connections are active, max_p increases significantly, causing the RED algorithm to become more aggressive. When only 8 connections are active, max_p decreases, causing the RED algorithm to become less aggressive.

3.4 Congestion Notification via Packet Drop

The previous experiments examine the use of early detection in its "purest" form where congestion notification is free and causes no packet loss. Without support for ECN, RED must resort to dropping a packet to signal congestion. This leads to an interesting optimization problem where the

RED queue must pick a max_p value which minimizes the sum of packet drops due to early detection and packet drops due to buffer overflow. With extremely large max_p values, packet loss rates are dominated by drops due to early detection while with extremely small max_p values, packet loss is mostly due to queue overflow.

In order to examine the performance of RED using packet drops, we repeated the above experiments. Figure 9 shows the loss rates for 32 and 64 connections in a drop-tail and in a RED queue for a range of max_p values. As the figure shows, RED only has a marginal impact on the packet loss rates observed. For small values of max_p, early detection is ineffective and the the loss rates in the RED queue approach loss rates in the drop-tail queue. As max_p is increased, loss rates decrease slightly since the RED queue is able to send congestion notification back to the sources in time to prevent continual buffer overflow. Finally, as max_p becomes large, the RED queue, in fact, causes a slight *increase* in packet loss rates over drop-tail queues. Note that the value of max_p that minimizes the loss rates is different in Figure 9(a) and Figure 9(b). As more connections are added, the optimal value of max_p increases.

The use of packet drops as a means for congestion notification fundamentally limits the effectiveness of active queue management. Steady-state analysis of the TCP congestion avoidance algorithm [7, 15, 17, 18, 20] gives some insight as to why this is the case. Such analysis has shown that given random packet loss at constant probability p, the upper bound on the bandwidth a TCP connection sees can be estimated as:

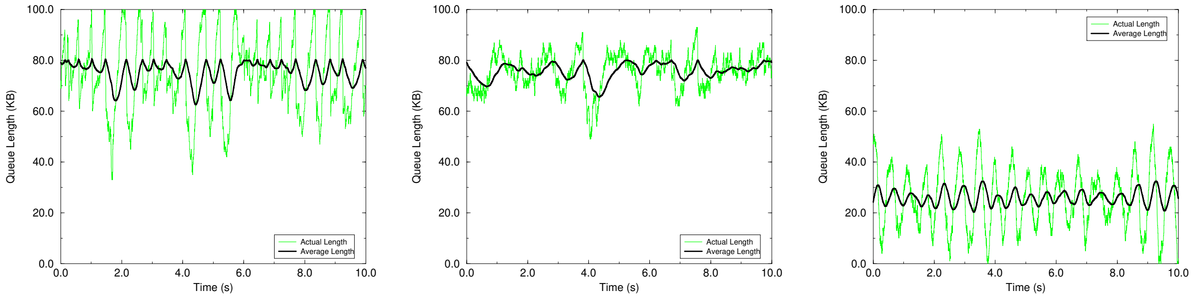
$$BW < \frac{MSS}{RTT} \frac{C}{\sqrt{p}} \tag{1}$$

where MSS is the segment size, RTT is the round-trip time, and C is a constant. Using this model, we can approximate packet loss rates over a single bottleneck link of L Mbs for a fixed number of connections N. In this situation, the bandwidth delivered to each individual connection is approximately the link bandwidth divided by the number of connections (L/N). By substituting this into the previous equation and solving for p, we obtain

$$p < \left(\frac{N}{L} \frac{MSS * C}{RTT}\right)^2 \tag{2}$$

As the equation shows, when all of the connections are using the TCP congestion avoidance algorithm, the upper bound on the packet loss rate quadratically increases with the number of connections present. Intuitively, this can be shown using an idealized example. Suppose we have two identical networks with bandwidth-delay products of $64KB$ from a given pair of end points. In one network, we have 4 identical connections while in another we have 8 identical connections going across the network. Given fair sharing amongst the connections, the congestion windows of each connection will approximately be the bandwidth-delay product divided by the number of connections present. For 4 connections, each connection will have congestion windows which oscillate near $16KB$. Assuming normal TCP windowing and a segment size of $1KB$, an individual connection in this network will typically build its congestion window up to around 16 packets, receive congestion notification in the form of a lost packet, back off its window to about 8 packets and then slowly build its congestion window back up at a rate of one packet per round-trip time. Given this behavior, the loss rate across all connections in this idealized model would then be approximately 4 packet drops every 8 round-trip times or 0.5 packets/round-trip time. Similarly, using the same idealized model, it can be shown that when 8 connections are present, losses occur at a rate of 2.0 packets/round-trip time, a quadratic increase.

Because the derivation of Equation (2) is based on idealized scenarios, the actual loss rates do not quite vary quadratically with the number of connections. From the drop-tail experiments in Figure 9, the loss rates observed show a dependence on the number of connections which is



(a) $max_p = 0.16$ (b) $max_p = 0.063$ (c) $max_p = 1.00$

Figure 10: Queue length plots of RED over max_p

somewhere between linear and quadratic. This is partially due to the fact that connections can experience retransmission timeouts causing them to send less packets than the equation predicts. Still, since packet loss rates increase rapidly as networks become more congested, it is necessary to decouple packet loss from congestion notification through the use of ECN in order to prevent congestion collapse.

In lieu of explicit congestion notification, Equation (2) provides some insight on how to reduce loss rates. In particular, decreasing the segment size, increasing the bottleneck bandwidth, and increasing the round-trip times through the use of additional buffers [24] can slow the rate of congestion notification considerably and thus, decrease the amount of packet loss observed. Note that the dependence of packet-loss rates on round-trip times explains why RED queues, which attempt to reduce network queueing delays, can potentially increase packet-loss rates (as observed in Figure 9). To examine this particular effect, Figure 10 plots the queue lengths of the congested RED queue over several values of max_p in the experiment with 32 connections. When max_p is small as in Figure 10(a), early detection is ineffective and the behavior of the RED queue approaches that of a drop-tail queue. Figure 10(b) shows the queue length plot for the max_p value which minimizes packet loss (0.063). In this case, the RED algorithm is aggressive enough to eliminate packet loss due to buffer overflow while keeping the average queue length as close to the size of the queue as possible. Finally, as max_p increases even further as shown in Figure 10(c), RED becomes more aggressive in its early detection leading to a drop in the average queue length. Consequently, the round-trip times seen by TCP connections also drop. Because packet-loss rates increase with decreasing round-trip times (see Equation (2)), this causes the loss rates in the RED queue to eventually exceed that of the drop-tail queue.

4 End-Host Congestion Control

While properly designed active queue-management mechanisms, such as adaptive RED, can help reduce packet losses, such techniques alone cannot guarantee low-loss rates. Active queue management must be combined with intelligent end-host congestion control in order to obtain high utilization with a minimal amount of packet loss. In this section, we examine several weaknesses in TCP's congestion control algorithm which become conspicuous under heavy load and can cause high loss rates even in the presence of active queue management. In light of this study, we propose and experiment with possible modifications to TCP's windowing mechanism which alleviate these problems.

```
(1) closecwnd()
        if (cwnd == 1)
                sub_scale = sub_scale * 2;
                counter = int(sub_scale*RTT/timer_interval) + 1;
        else normal_tcp_closecwnd();
(2) send()
        if (cwnd == 1)
                if (counter ≤ 0)
                        send_next_segment;
                        counter = int(sub_scale*RTT/timer_interval) + 1;
                return;
        else normal_tcp_send();
(3) opencwnd()
        if (cwnd==1 && sub_scale > 1)
                sub_scale = sub_scale / 2;
                if (sub_scale < 1) sub_scale = 1;
                return;
        normal_tcp_opencwnd();
(4) Every timer_interval
        counter = counter - 1;
```

Figure 11: SUBTCP algorithm

4.1 Adjusting the Minimum Transmission Rate

One of the limitations of TCP's congestion control mechanism is that in normal operation, the minimum transmission rate of a TCP sender is one segment per round-trip time [4]. This does not appear to be a reason for serious concern at first glance. However, in many typical operating environments [3] this limitation in TCP's congestion control mechanism may lead to a high packet-loss rate [19]. TCP does, in fact, have a mechanism which allows it to transmit at a rate lower than one segment per round-trip time. When a single segment is transmitted and continually dropped, TCP doubles its retransmission interval until the segment has been delivered and acknowledged. However, upon a single successful transmission, TCP resets this timer and resumes transmitting at a rate of one segment per round-trip time. This, in effect, increases the transmission rate exponentially. If a large number of connections increase their transmission rates in this manner, the combined effect can cause large fluctuations in the offered load.

One way to alleviate this problem is to decrease the exponential back-off interval when a packet is successfully transmitted rather than resetting the value. This prevents the source from automatically ramping up its transmission rate as soon as it receives a single acknowledgment after a retransmission timeout. Figure 11 shows a simple algorithm for doing so. In this case, instead of resetting back-off interval after a successful transmission, it is halved. In the rest of the paper, this variation of TCP is referred to as SUBTCP. TCP's minimum transmission rate is also directly determined by the segment size and the minimum window size used. Smaller segment and window

[3] For example, this situation may arise when 100 TCP connections each with a segment size of 1Kbytes and an RTT of 200ms share a T1 link. This is a very typical scenario in many ISP networks.

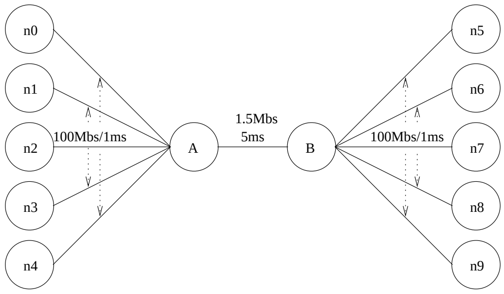
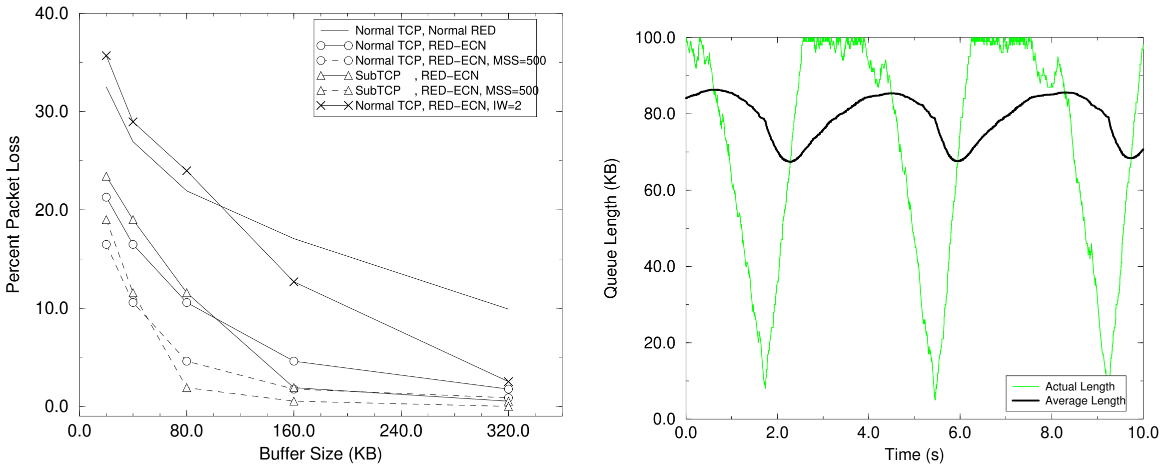


Figure 12: Network Topology



(a) Packet-loss rates (b) Queue plot of normal TCP with RED-ECN

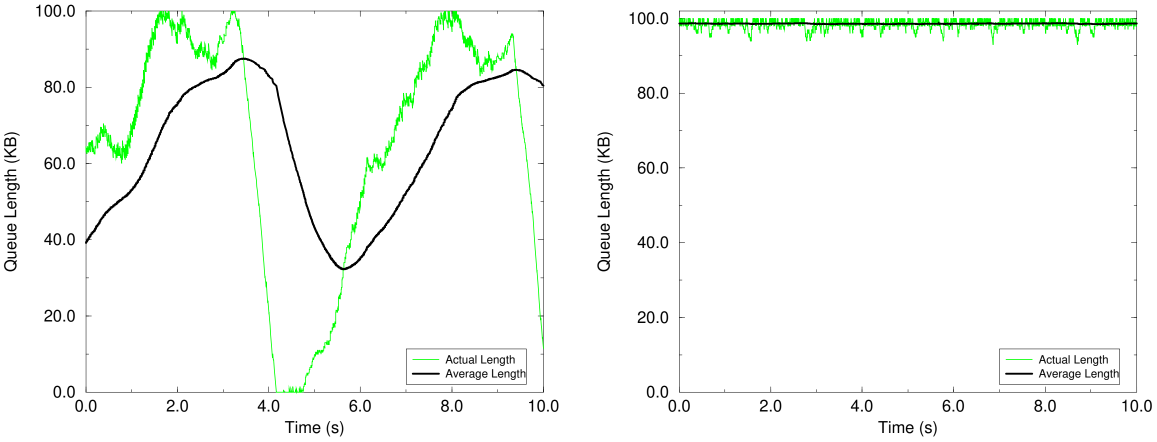
Figure 13: Minimum sending rates and the performance of TCP

sizes translate into lower minimum transmission rates.

To understand the impact that the SUBTCP modifications, the segment size, and the minimum window size have on packet-loss rates in congested networks, we examine a number scenarios in the network shown in Figure 12. In this network, 100 connections are run from nodes $n0$–$n4$ through the bottleneck 1.5 Mbs link between node A and node B to nodes $n5$–$n9$.[4] For most of the experiments in this section, we make early detection conservative by fixing max_p to 0.016, in order to isolate the impact end-host modifications have on performance. Figure 13(a) shows the packet-loss rates observed by TCP and SUBTCP for different segment sizes and minimum windows. As the figure shows, the use of the smaller segment size of 500 bytes significantly improves loss rates over the use of the larger 1000 byte segment size. In addition, the figure also shows that the SUBTCP modifications have a very modest impact on the loss rates. Finally, we observe that doubling TCP's minimum window leads to a significant increase in the amount of packet loss observed. For a better understanding of the above observations, we examine the queue-length plots of the bottleneck link for the various experiments. Figure 13(b) shows the queue-length plot of the experiment using TCP with RED-ECN queues[5]. The max_{th} used in the experiment was 80 KB. The

[4] For the RED queues, max_{th} is used as the measure of buffer size in an attempt to more fairly compare drop-tail and RED queues. The actual queue size is set at $(1.2{*}max_{th})$ while min_{th} is set at $(0.25{*}max_{th})$.

[5] The queue length plot of normal TCP over normal RED queues shows results similar to Figure 3(b).



(a) SUBTCP with MSS=500 (b) TCP with minimum window = 2

Figure 14: Queue plots for decreased segment size and increased window size

figure shows that the queue quickly fills up soon after congestion notification stops being delivered. This is due to the fact that TCP ramps up its transmission rate too quickly upon successfully sending its segments, causing large queue-length fluctuations which defeat the early detection mechanism and induce packet loss.

Figure 14(a) shows the queue-length plot using the smaller segment size and the SUBTCP modifications. As the figure shows, the SUBTCP modifications slow the increase in transmission rates of individual connections and thus help reduce the amount of queue fluctuations and packet loss. Note that the SUBTCP modifications alone are not enough to allow early detection to work. While SUBTCP can reduce the minimum transmission rates of the sources, it still allows a multiplicative increase in transmission rates. The resulting traffic pattern defeats early detection causing packet loss to occur. Additional modifications to address this problem are described in Section 4.2.

Figure 14(b) shows the queue-length plot of normal TCP over RED-ECN queues using an minimum window of two segments. We note that recent work in the IETF [9] has recommended increasing the initial window size of TCP connections to at least two segments. As shown in the trace, using the large minimum window defeats the early detection mechanism of RED as well as the congestion control mechanism of TCP. The queue remains fully occupied through the duration of the experiment. Thus, while a large minimum window may be desirable in unloaded networks where it can reduce transfer latencies considerably, it must be used carefully when the network is congested.

4.2 Adjusting Linear Increase

In steady state, TCP uses its congestion-avoidance algorithm to slowly increase its transmission rate. In this phase, the congestion window of a connection increases linearly by one segment per round-trip time. During times of congestion when a large number of connections are competing for bandwidth, the window size of each connection is small. Unfortunately, for small windows, linear increase is a misnomer since increasing the window size by a segment can have a non-linear impact on the connection's transmission rate. For example, when a connection has a congestion window of 1, it doubles its sending rate when it increments its window by a segment. When a large number of connections with small windows are aggregated, the network sees traffic which multiplicatively increases and decreases. This causes rapid fluctuations in queue lengths, periods

```
opencwnd()
    if (cwnd==1 && sub_scale > 1)
        sub_scale = sub_scale - scaled_increase;
        if (sub_scale < 1) sub_scale = 1;
    else normal_tcp_opencwnd();
```

Figure 15: Scaled Sub-linear SUBTCP algorithm

```
opencwnd()
    effective_cwnd = (cwnd/sub_scale);
    tcp_increase = 1/effective_cwnd;
    bw_increase = effective_cwnd*bandwidth_percent;
    new_wnd = effective_cwnd + min(tcp_increase, bw_increase);
    if (new_wnd < 1)
        cwnd = 1;
        sub_scale = 1/new_wnd;
    else
        cwnd = new_wnd;
        sub_scale = 1;
```

Figure 16: Bandwidth-based SUBTCP algorithm

of high utilization and packet loss followed by periods of underutilization. Such traffic patterns also defeat early detection mechanisms such as RED because large queues can accumulate over very short time periods.

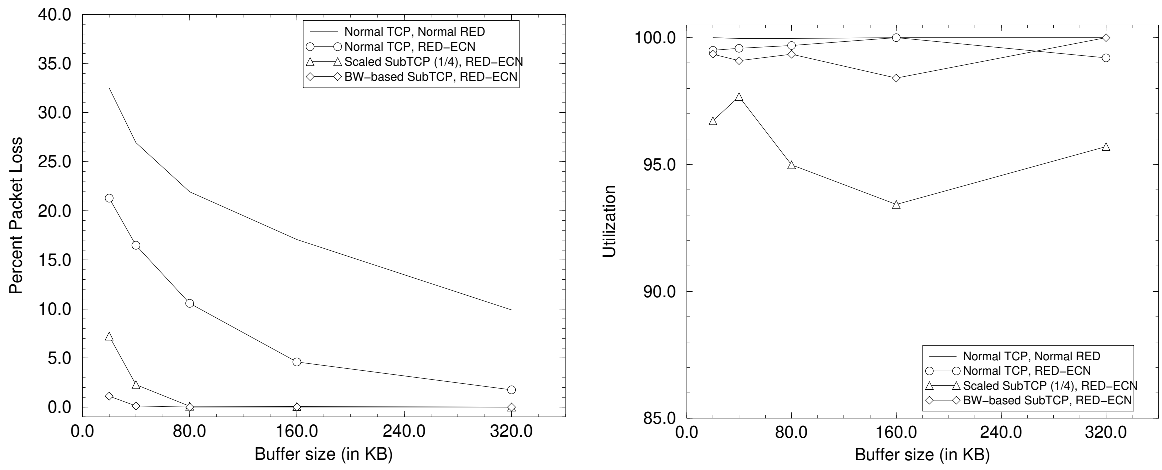
In this section, we examine how the linear increase mechanims can be modified to work better under heavy congestion. We consider two, more conservative, window increase algorithms: (1) scaled linear increase and (2) bandwidth-based linear increase. The idea behind the scaled increase is to increase the congestion window by a fixed fractional amount when the congestion window is small. This is a heuristic much like the current linear-increase algorithm. While it can certainly alleviate some of the problems seen with standard linear increase, fixed increases in general have the problem of either being too conservative when the network is uncongested or being too aggressive when the network is congested. Figure 15 shows the scaled linear-increase algorithm used. In our experiments, we use the scaled increases to slow the increase in transmission rates when the congestion window is effectively below one. Thus, when the network is not congested, the source simply behaves as a normal TCP source.

In addition to the scaled linear-increase algorithm, we also experimented with a bandwidth-based linear-increase algorithm. The motivation of the bandwidth-based algorithm is rather intuitive. Assume that the bottleneck router has sufficient number of buffers to absorb incoming traffic with aggregate throughput $X\%$ higher than its link bandwidth for a duration of about a round-trip time. This is the time it takes for congestion notification to reflect itself back to the bottleneck link. Then, if each source only increases its transmission rate by $X\%$ every round-trip time, the network can ensure a minimal amount of packet loss. Bandwidth-based increases inherently depend on the round-trip times of each of the connections. This is because it takes at least a round-trip time for congestion notification to have an impact on the offered load seen by the router. Bandwidth-based increases also depend on the amount of buffering at the bottleneck link. If we add more buffers at the bottleneck link, the TCP sources can be more aggressive in ramping up their transmission rates. On the other hand, increased buffering can also prevent traffic sources from ramping up their transmission rates by increasing the round-trip time and thus the latency in delivering congestion feedback. Even with bandwidth-based increases, there is still a small likelihood of buffer overflow. One reason is that in the case of RED queues, deterministic congestion notification is triggered only when the weighted average exceeds max_{th}. The actual queue length can often be larger than the average queue length especially when the offered load is steadily increasing. Another reason packet loss occurs is that TCP traffic exhibits short-term burstiness which can make its offered load appear to be much larger than it actually is over short time periods [27]. Such burstiness can cause buffer overflow even when the overall offered load is below the bottleneck link capacity. Still, the key advantage of using bandwidth-based linear increases is that the behavior of its aggregate traffic is mostly independent of the number of connections present. The router thus sees traffic which increases at a fairly fixed rate regardless of the amount of congestion in the network. Controlling the behavior of aggregate traffic allows active queue-management schemes to work as intended.

Figure 16 shows the bandwidth-based linear-increase algorithm used in our experiments. As the algorithm shows, the window increase used is calculated as the minimum of TCP's current linear increase and a calculated bandwidth-based increase. We note that one of the key disadvantage of the bandwidth-based increase algorithm is that it falls in a class of algorithms which has been theoretically shown to not provide responsive max-min fair sharing [3, 14]. While fairness is a concern, even with TCP's current congestion control algorithm, fairness between connections has already been shown to be poor in times of congestion and among connections with varying round-trip times [10, 19]. In addition, the idealized model used in [3] assumes that congestion notification is given to all sources when the bottleneck resource becomes saturated. With more intelligent queueing algorithms such as RED and FRED [9, 16] that deliver congestion notification preferentially to higher bandwidth flows, it may be possible for a larger class of increase/decrease algorithms to quickly converge to max-min fair sharing.

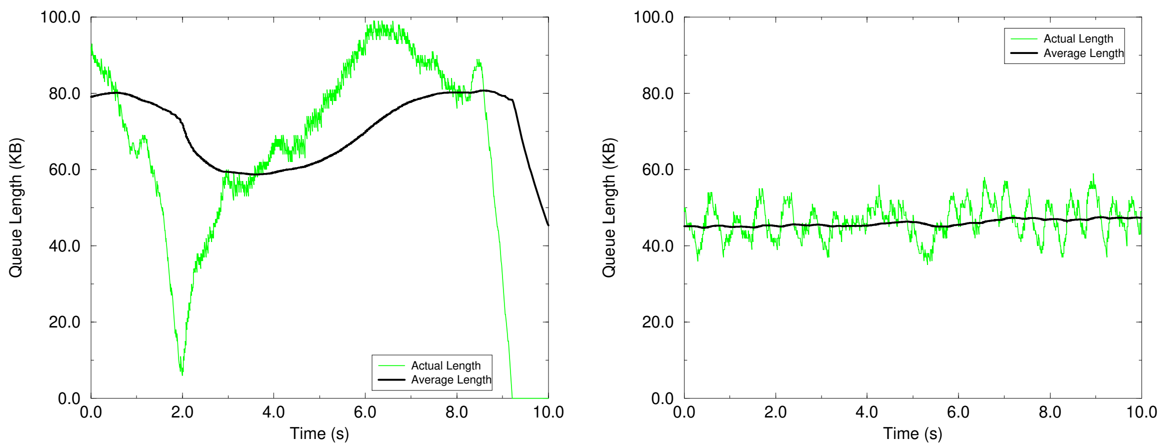
To evaluate these modified increase algorithms, we reran the previous experiments in order to evaluate their performance. Figure 17 shows the loss rates and the link utilization observed. For the fixed scaled increases, the graphs show the results using a scaling factor of $\frac{1}{4}$. For the bandwidth-based linear-increase algorithm, we set the bandwidth increase steps to 1% in order to prevent packet loss in the network shown in Figure 12. Figure 17(a) shows the loss rates observed using the various TCP schemes. As can be observed from the graph, sublinear increases greatly reduce the amount of packet loss. In fact, for smaller buffer sizes, the improvements in loss rates is as high as *several* orders of magnitude. The bandwidth-based algorithm, in particular, provides extremely low loss rates under heavy congestion even when a small amount of buffering is present. Figure 17(b) shows the link utilization observed across all schemes. While the scaled linear-increase algorithm provides low losses, it is unable to sustain full link utilization. The bandwidth-based algorithm, on the other hand, maintains high link utilizations across all buffer sizes.

In order to understand why the scaled linear-increase algorithm sees lower link utilization and sometimes higher packet loss rates than the bandwidth-based algorithm, we examine the queue-length plot of the bottleneck queue. Figure 18(a) plots the queue lengths from the experiment using a max_{th} of 80 KB. The traces show that the use of sublinear increases makes aggregate TCP less



(a) Packet loss rates (b) Bottleneck link utilization

Figure 17: Performance of modified linear-increase algorithms

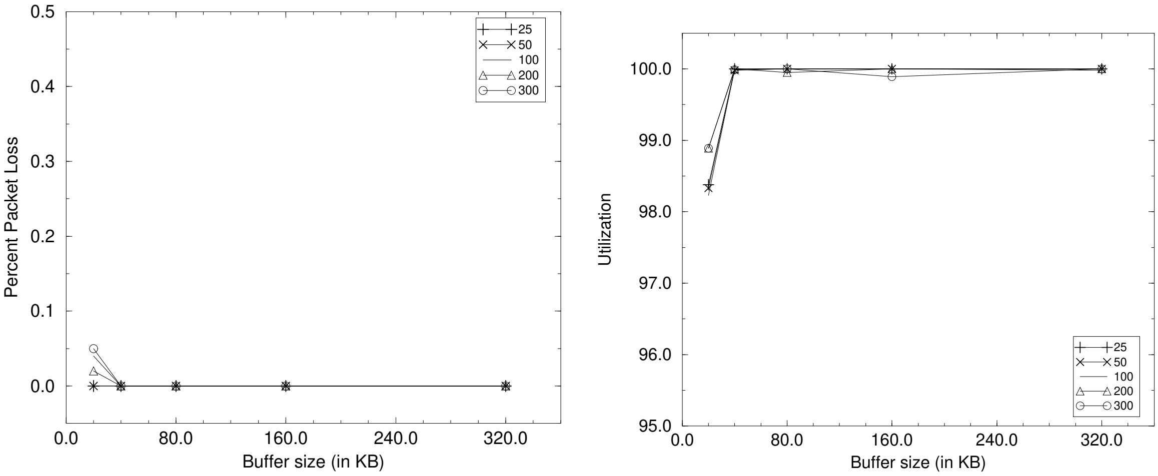


(a) Scaled increases (scale $= \frac{1}{4}$) (b) Bandwidth-based increases

Figure 18: Sublinear SUBTCP Performance

aggressive, thus preventing sustained periods of packet loss observed in the previous experiments shown in Figure 14. However, they are still aggressive enough to defeat the early detection mechanism. It takes a sustained period of congestion notification when the average queue length exceeds max_{th} ($t = 7.0s$) for the offered load to be reduced sufficiently below link capacity to clear the queue. However, as shown earlier in Section 3.2, sustained congestion notification can be detrimental to link utilization since it can potentially cause too many sources to back-off their transmission rates. As the queue plot shows in Figure 18(a), after max_{th} is triggered, the bottleneck link subsequently becomes underutilized ($t = 9.0s$).

There are a couple of ways to alleviate this problem. One would be to make the scaled increases even smaller so that static early detection has a chance to signal the end-hosts in time to prevent max_{th} from being triggered. Another way to improve performance is to simply make the early detection mechanism more aggressive as described in Section 3. Additional experiments have shown



(a) Packet loss rates

(b) Bottleneck link utilization

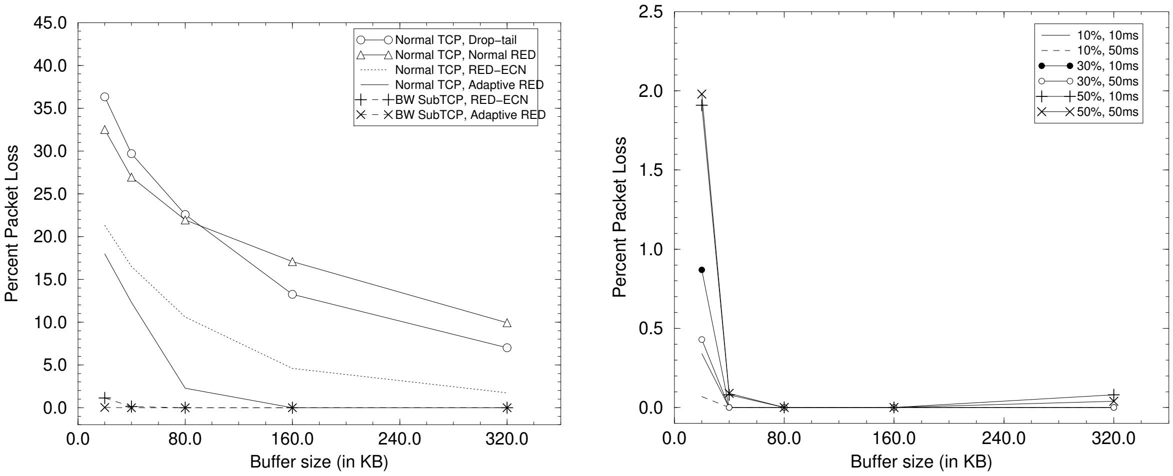
Figure 19: Performance across traffic load

that both techniques can be effective in improving the performance of scaled increases. In contrast to the scaled linear increase, Figure 18(b) shows the queue-length plot when bandwidth-based increase is used. In this case, the aggregate traffic fluctuates much less as compared to the previous case. Consequently, the statically configured bottleneck RED queue is able signal end-hosts at a rate which can prevent packet loss and deterministic congestion notification. Thus, the network maintains a high degree of efficiency.

5 Tuning for Optimal Performance

The previous two sections have shown how individually, active queue-management and end-host mechanisms can be used to significantly reduce loss in the network. When used together, they form a synergistic combination which can allow the network to achieve extremely high efficiency even in times of heavy congestion. Figure 19 shows the loss rates and bottleneck link utilizations over a range of workloads using the bandwidth-based SUBTCP algorithm with adaptive RED queues in the network. In these experiments, we use the same topology as in Figure 12 and vary the number of connections going across the $1.5Mbs$ link from 25 to 300. As the figure shows, the packet-loss rates remain extremely low while the bottleneck link utilizations remain remarkably high across all experiments.

In order to compare the improvements against other schemes, we examine the packet loss rates using a traffic load of 100 connections. Figure 20(a) plots the loss rates for a range of end-host and queue-management schemes. The figure shows the performance of standard TCP using both drop-tail and RED queues as well as the performance of TCP using ECN with both a static RED-ECN queue and an adaptive RED-ECN queue. In addition, the figure plots the performance of the bandwidth-based SUBTCP algorithm over both static and adaptive RED-ECN queues. The graph shows that decoupling congestion notification from packet drop through the use of ECN reduces loss rates significantly. In addition, the use of the adaptive RED algorithm along with ECN allows the network to achieve extremely low loss rates when sufficient buffering is provided. Finally, the



(a) Comparison between schemes (b) Round-trip time and parameter sensitivity

Figure 20: Performance comparisons

graph shows that when both the adaptive RED and SUBTCP modifications are used together, they can provide optimal performance regardless of the amount of buffering in the network. This allows the network to maintain high degrees of efficiency without having to incur large queueing delays through the use of additional network buffers [19, 24].

The previous experiments fixed both the round-trip times and the percent bandwidth increase used (1 %). Since the performance of the proposed congestion control mechanisms have an inherent dependence on both, we ran several experiments which varied them. Figure 20(b) plots the loss rates using 100 connections when the percent bandwidth increase used is 10 %, 30 %, and 50 %. The experiments also vary the transmission delay across the bottleneck link from $10ms$ to $50ms$, thus considerably increasing the round-trip time. As the figure shows, as each source is allowed to increase its transmission rate more quickly, loss rates slowly rise as the burstiness seen at the router increases. However, even with larger bandwidth increase steps, loss rates still remain relatively low compared to other schemes. The figure also shows that an increased round-trip time has little impact on performance.

6 Conclusion and Future Work

This paper has shown how active queue management and end-host congestion control algorithms can be designed to effectively eliminate packet loss in congested networks. In particular, an adaptive RED mechanism which is cognizant of the number of active connections and the use of bandwidth-based linear increases can provide significant benefits in terms of decreasing packet loss and increasing network utilization. We are currently working on extending both mechanisms. In particular, we are examining ways of methodically improving the adaptiveness of the RED algorithm. This paper presents one specific algorithm for tailoring RED parameters to the input traffic. There are many other potential alternatives for doing so. For example, the RED queue can actually keep track of the number of active connections and change its aggressiveness accordingly. This allows the queue to adapt immediately whenever a large number of connections suddenly appear or disappear as is the case with web traffic, in particular. Another mechanism would be to have the RED queue infer the number of connections present by the rate that the average queue length changes, then adapt its parameters accordingly. The aggressiveness of RED could also be changed based on traffic history and the current time of day, week, or year. It may also be possible to adapt other RED parameters instead of max_p to optimize performance. For example, one could adaptively change inter-packet drop probabilities or the RED threshold values depending on the input traffic. Finally, we are examining ways of applying the adaptive RED algorithm to an enhanced ToS-enabled version of RED (ERED) [5, 6] in order to improve its performance, as well.

We are also examining ways of improving end-host congestion control algorithms. While bandwidth-based increases provide end-hosts with an upper bound on how aggressively they can ramp up their transmission rates, it is often desirable for a source to change its transmission rate more slowly or not at all when nearing the congestion point [2, 25, 26] in order to avoid oscillations inherent in TCP's windowing algorithm. We are currently investigating how to incorporate such techniques into the bandwidth-based increase algorithm. We are also exploring additional mechanisms to improve the fairness of the windowing algorithms. One of the advantages of TCP congestion avoidance is that it inherently gives an advantage to low-bandwidth flows which allows the algorithm to converge quickly to max-min fairness. Using additional end-host mechanisms and router fairness mechanisms such as FRED [9, 16], we are examining ways of obtaining the benefits of bandwidth-based increases while maintaining fairness among connections.

References

[1] R. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. *Internet Draft draft-irtf-e2e-queue-mgt-00.txt*, March 1997.

[2] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of ACM SIGCOMM*, pages 24–35, October 1994.

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

[4] J. Crowcroft. TCP very slow start and freeway avoidance. *end2end-interest mailing list*, September 1995.

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin. Adaptive Packet Marking for Providing Differentiated Services in the Internet. *UM-CSE-TR-347-97 and IBM RC 21013*, October 1997.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin. Understanding TCP Dynamics in an Integrated Services Internet. In *Proc. of NOSSDAV '97*, May 1997.

[7] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-way Traffic. *Computer Communication Review*, 21(5), October 1991.

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

[9] S. Floyd, M. Allman, and C. Partridge. Increasing TCP's Initial Window. *Internet Draft draft-floyd-incr-init-win-00.txt*, August 1997.

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

[11] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *ACM/IEEE Transactions on Networking*, 1(4):397–413, August 1993.

[12] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, pages 314–329, August 1988.

[13] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. *end2end-interest mailing list (ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt)*, April 1990.

[14] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. *DEC Research Report TR-301*, Sept. 1984.

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

[16] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proc. of ACM SIGCOMM*, September 1997.

[17] M. Mathis and J. Semke and J. Mahdavi and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27(1), July 1997.

[18] S. McCanne and S. Floyd. http://www-nrg.ee.lbl.gov/ns/. ns-LBNL Network Simulator, 1996.

[19] R. Morris. TCP Behavior with Many Flows. In *Proc. IEEE International Conference on Network Protocols*, October 1997.

[20] T. Ott, J. Kemperman, and M. Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. *ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps*, August 1996.

[21] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. of ACM SIGCOMM*, September 1997.

[22] K. K. Ramakrishnan and S. Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IPv6 and to TCP. *Internet Draft draft-kksjf-ecn-00.txt*, November 1997.

[23] K. 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.

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

[25] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *Computer Communication Review*, 21(1):32–43, January 1991.

[26] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in 4.3-Tahoe BSD TCP Congestion Control Algorithm. *Computer Communication Review*, 22(2):9–16, April 1992.

[27] L. Zhang, S. Shenker, and D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of ACM SIGCOMM*, pages 133–148, August 1991.