A Self-Configuring RED Gateway

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

†Department of EECS
University of Michigan
Ann Arbor, MI 48105
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. The IETF has addressed this problem by advocating the deployment of active queue management mechanisms, such as RED, in the network. While RED can potentially improve packet loss rates, we show that its effectiveness is highly dependent upon its operating parameters. In fact, in cases where these parameters do not match the requirements of the network load, the performance of the RED gateway can approach that of a traditional drop-tail gateway. To alleviate this problem, we propose and experiment with a self-configuring active queue management mechanism which can significantly reduce loss rates across congested links. When used in the network, this mechanism can effectively reduce packet loss while maintaining high link utilizations under the most difficult scenarios.

Keywords: Congestion control, Internet, TCP, RED, queue management

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 [10]. 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 this high packet-loss rate is the failure of the network to provide early congestion notification to sources. This has led the IETF to recommend the use of active queue management in Internet routers [1, 17]. There have been several active queue management algorithms which have been proposed including random drop [9, 16], early packet discard [24], and early random drop [11]. The most prominent and widely studied active queue management scheme is Random Early Detection (RED) [1, 8]. RED gateways alleviate many of the problems found in other active queue management algorithms in that they can prevent global synchronization, reduce packet loss rates, and minimize biases against bursty sources using a simple, low-overhead algorithm.

While RED can certainly outperform traditional drop-tail queues, our experiments show that it is difficult to parameterize RED queues to perform well under different congestion scenarios. This paper identifies two key problems with using RED queues. The first is that congestion notification does not directly depend on the number of connections multiplexed across the link. Section 3 shows that the effectiveness of early detection is critically dependent upon the rate at which congestion notification is provided to the sources and the network load. When congestion notification is aggressive, packet loss can be avoided, but at the expense of low overall link utilization. On the other hand, when the rate of congestion notification is not sufficiently aggressive, link utilization remains high but the increased packet loss results in a drop in goodput and an increase in wasted resources. The second problem with RED is that its benefits diminish significantly when packet loss is used as a means of congestion notification. Since packet loss rates can potentially increase quadratically with the number of TCP connections multiplexed, decoupling congestion notification from packet loss must be done in order to ensure low loss rates in times of congestion. By using ECN in conjunction with a queue management mechanism which is cognizant to the number of connections multiplexed, Sections 4 and 5 describe the design and implementation of an auto-configuration mechanism that determines suitable operating parameters for a RED queue depending on the traffic load. When used, this mechanism helps RED queues minimize packet loss while maximizing link utilization. A key feature of this mechanism is that it does not require per-flow accounting and retains the main advantages of RED. This is in contrast with other approaches which introduce additional complexity to the queue management scheme in return for additional functionality. Such schemes add either per-flow queueing [25] or per-flow accounting [14] in order to improve performance. Because of its simplicity and low overhead, the modifications proposed in this paper are particularly well-suited for managing traffic through large backbone routers where per-flow accounting and queueing may not be available. Given the performance improvements that this mechanism can provide, 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 sending rate by limiting the number of unacknowledged packets the sender can

Parameter	Function
min_{th}	Threshold which the average queue length must exceed before any dropping or marking is done.
max_{th}	Threshold which the average queue length must exceed before all packets are dropped or marked.
max_p	Maximum marking probability. Determines how aggressively the queue drops or marks packets when congestion occurs.
w_q	Weight for updating average queue length. Large values increase the weight of more recent queue length data
$f(t)$	Inter-packet drop probability function

Table 1: RED parameters

have outstanding in the network. When a connection initially 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 [10]. In this algorithm, TCP keeps track of a threshold value (`ssthresh`) which is a conservative approximation of the window size that 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 the 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 duplicate acknowledgments from the receiver [12]. Upon receiving a small number of duplicate acknowledgments, TCP infers that a packet loss has occurred and immediately cuts its transmission rate in half 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.

One problem with the TCP congestion control algorithm over current networks is that TCP sources back off their rates only after detecting packet loss caused by 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 transmit 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 queues overflow and packet loss occur. One of the more promising active queue management schemes being proposed for deployment in the network is RED (Random Early Detection) [7]. A RED gateway maintains an exponentially weighted moving

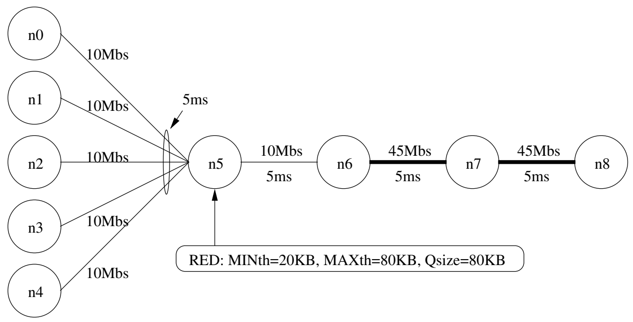


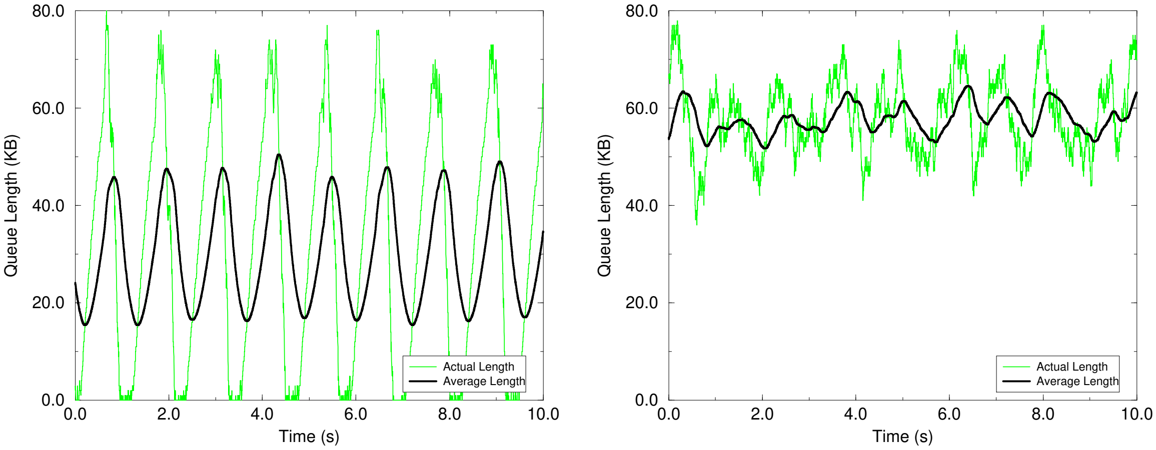
Figure 1: Network Topology

average of the queue length as a means for detecting congestion. When the average queue length exceeds a minimum threshold (min_{th}), packets are dropped or marked with an explicit congestion notification (ECN) bit [7, 22, 23] at random using a calculated probability. When a TCP sender receives congestion notification in the form of an ECN mark, it halves its congestion window 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 upon, 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. In our experiments, the aggressiveness of RED is varied by changing max_p, RED's initial drop probability. While it is possible to vary any of RED's parameters (as shown in Table 1), we vary max_p since it directly impacts the aggressiveness of the early detection mechanism. One could potentially change $f(t)$ and w_q, the inter-drop probability function and the queue weight respectively, however, doing so could potentially impact RED's fairness to bursty connections as well as its ability to avoid global synchronization.

3 Weaknesses in Active Queue Management

One of the inherent weaknesses of RED is that congestion notification does not directly depend upon the number of connections multiplexed over the link. In heavily-congested networks, in order to avoid packet loss due to buffer overflow, congestion notification must be signaled to a sufficiently large number of sources so that the offered load is reduced considerably. On the other hand, care must be taken to prevent excessive congestion notification since this can result in underutilization of the network bandwidth.

For example, consider a bottleneck link of capacity $10Mbs$ which is equally shared amongst several connections. When 100 TCP connections share this link, sending congestion notification to one connection reduces the offered load to $9.95Mbs$. On the other hand, when only 2 TCP connections share the link, sending congestion notification to one of them reduces the offered load to $7.5Mbs$. In general, when the bottleneck link is shared between N connections, a congestion notification signal to one connection reduces the offered load by a factor of $(1 - \frac{1}{2N})$. Hence, as N becomes large, the impact of individual congestion notifications decreases. In this situation, if the RED algorithm is not configured to be more aggressive, the



(a) 8 Connections

(b) 32 Connections

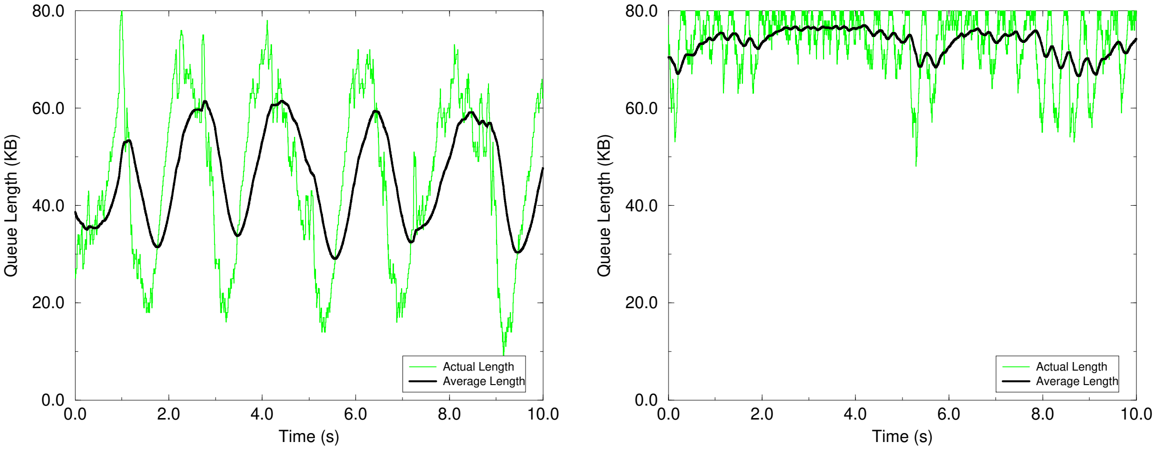
Figure 2: Aggressive early detection (max_p = 0.250)

RED queue can degenerate into a simple drop-tail queue. On the other hand, when N is small, the impact of individual congestion notifications is large. In this condition, if the RED algorithm is aggressive, underutilization can occur when many sources back off their transmission rates in response to the observed congestion. In the rest of this section, we quantify the impact of traffic load on RED and propose enhancements for optimizing its performance.

3.1 RED with ECN

In order to examine the impact of traffic load 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 these experiments. Note that each link is full-duplex, so acknowledgements flowing on the reverse path do not interfere with data packets flowing on the forward path. 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-enabled TCP. By using ECN, all packet losses from the RED queue can be attributed to buffer overflow. We first focus on the effects of congestion notification in the desired operating range for RED where the queue length is between the min_{th} and max_{th} thresholds. This is achieved by setting max_{th} to the queue limit, which and causes packet loss to occur whenever early detection is not effective. In these experiments, the aggressiveness of RED algorithm is controlled using max_p.

Figure 2(a,b) shows a plot of the queue-length at the link between $n5$ to $n6$ when the number of connections active is 8 and 32 respectively. As Figure 2(a) shows, when only 8 connections are active, aggressive early detection signals congestion notification to the sources at a rate which is too high, causing the offered load to be significantly smaller at times than the bottleneck link bandwidth. This causes periodic under-



(a) 8 connections (b) 32 connections

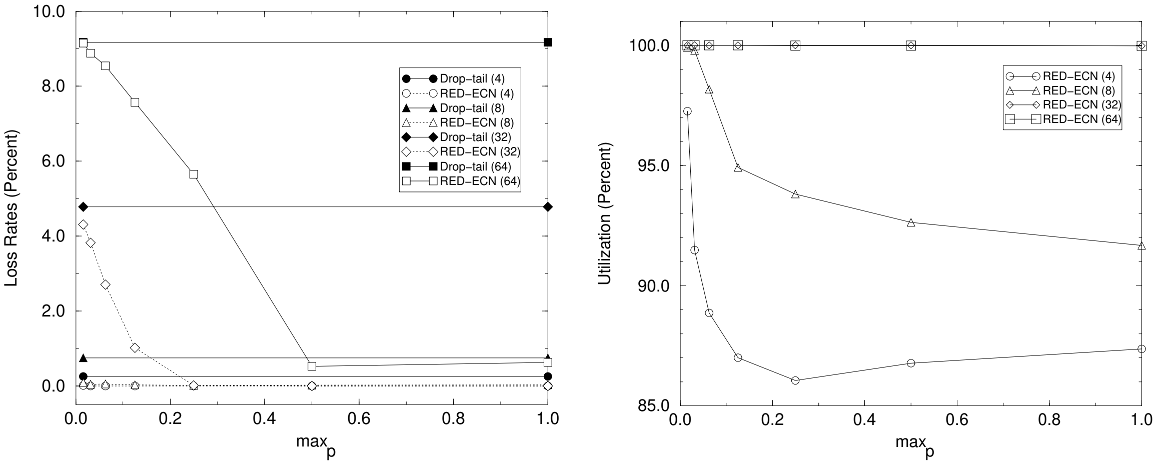
Figure 3: Conservative early detection (max_p = 0.016)

utilization 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(a,b) repeats the same set of experiments, but with a more conservative early detection setting. In contrast to Figure 2(a), Figure 3(a) shows that by using less aggressive early detection, the RED queue can maintain high link utilization while avoiding packet loss when the number of connections is small. However, when the number of connections is increased to 32, Figure 3(b) shows that conservative early detection does not deliver congestion notification to a 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 the 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 systematically 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

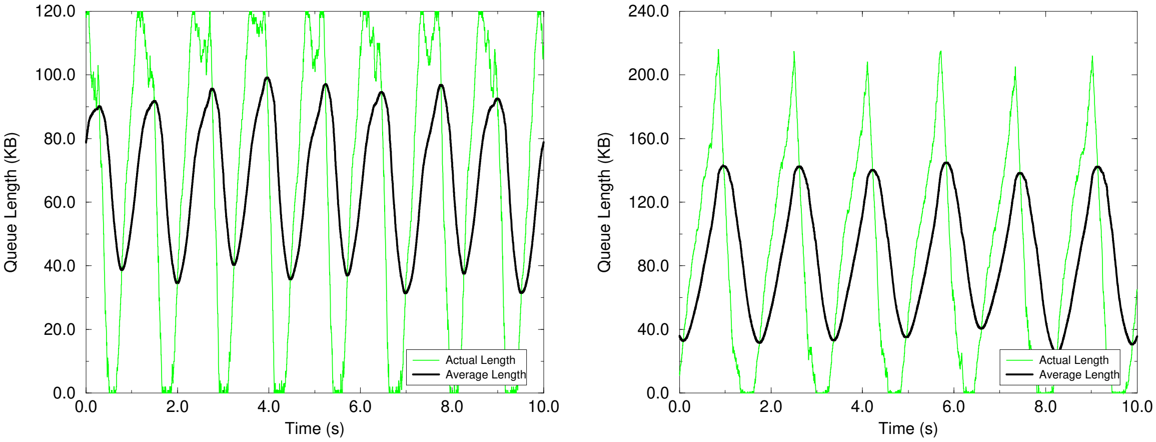
[1] In each experiment, connections are started within the first 10 seconds of simulation. After 100 seconds, both the loss rates and the link utilization for the bottleneck link are recorded for 100 seconds. The loss rate is calculated as the number of packets dropped by the queue divided by the total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

(b) Bottleneck link utilization

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



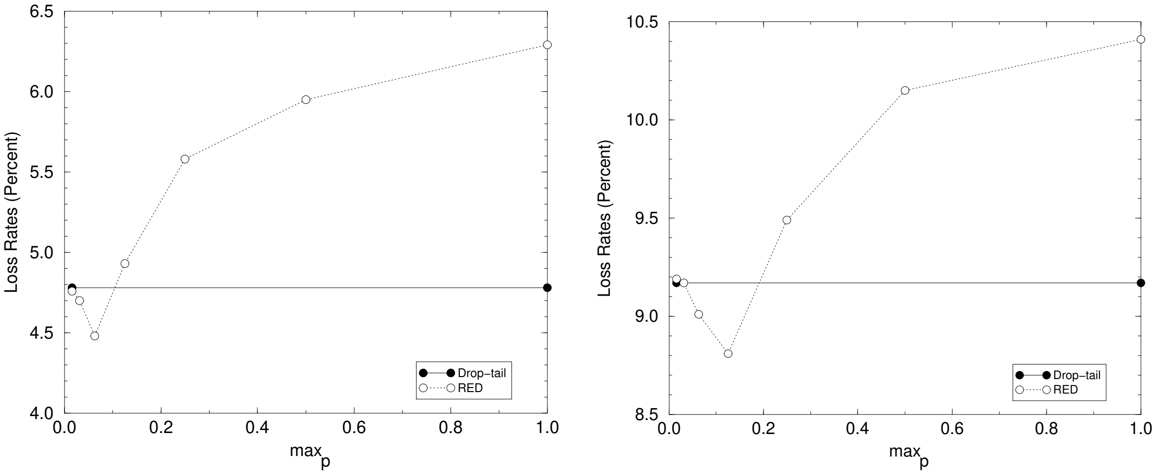
(a) queue size=$120KB$; max_{th}=$80KB$

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

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

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 values of max_p and 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.

In the previous experiments, 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, 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



(a) 32 connections (b) 64 connections

Figure 6: Impact of early detection aggressiveness on RED

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 either in the form of ECN or packet loss. Thus, as Figure 5 shows, after a sustained period of ECN marking and packet loss, the sources back off sufficiently 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, the buffer space required to reduce loss is substantial. Figure 5(b) plots the queue-length using a queue size limit of $240KB$. The figure shows that even though deterministic marking is enforced at an average queue length of $80KB$, the actual queue length can more than double before the sources have a chance to back off.

3.2 RED with Packet Drop

Experiments in the previous subsection examined the use of early detection in its "purest" form, where congestion notification is free and causes no packet loss. When support for ECN is not available, RED must resort to packet drops to signal congestion. This leads to an interesting optimization problem where the RED queue must pick a max_p value which minimizes the overall packet loss, including drops due to early detection and packet drops due to buffer overflow. When extremely large values of max_p are used, packet loss rates are dominated by drops due to early detection, while with extremely small values of max_p 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 6 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 has only a marginal impact on the packet loss rates observed. For small values of max_p, early detection is ineffective and 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 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 6(a) and Figure 6(b). As more connections are added, the optimal value of max_p increases.

Use of packet drops as a means for congestion notification fundamentally limits effectiveness of active queue management. Steady state analysis of the TCP congestion avoidance algorithm [6, 13, 15, 18, 20] provides 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 of a TCP connection can be estimated as:

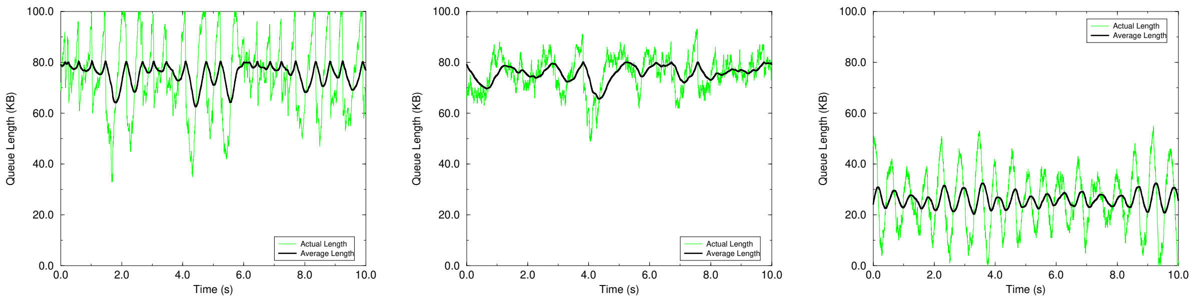
$$BW < \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{1}$$

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 connections use the TCP congestion avoidance algorithm, the upper bound on the packet loss rate increases quadratically 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 the other 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.

It should be noted that 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 6, the



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

Figure 7: Queue length plots of RED over max_p

observed loss rates show a dependence on the number of connections which is 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. One way would be to eliminate the quadratic dependence on loss rate by making TCP's linear increase mechanism more conservative. For example, by making the increase linear in terms of the sending rate, loss rates can be made to increase linearly with the number of connections[2]. Without changing end-host congestion control, Equation 2 shows that decreasing the segment size, increasing the bottleneck bandwidth, and increasing the round-trip times through the use of additional buffers [26] 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 time explains why RED queues, which attempt to reduce network queueing delays, can potentially increase packet loss rates (as observed in Figure 6). To examine this particular effect, Figure 7 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 7(a), early detection is ineffective and the behavior of the RED queue approaches that of a drop-tail queue. Figure 7(b) shows the queue length plot for the max_p value which minimizes packet loss (0.063). In this case, the RED algorithm is sufficiently aggressive 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 7(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.

[2] Note that such a scheme may impact the max-min fairness of linear increase and multiplicative decrease algorithms

```
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 / α;
        if ( Q(ave) > max_th && status != Above)
                status = Above;
                max_p = max_p * β;
```

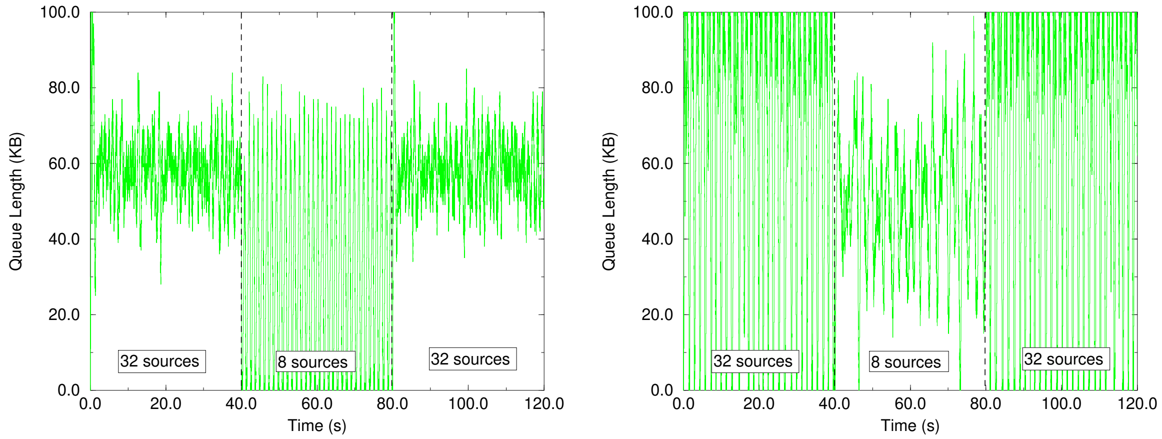
Figure 8: Adaptive RED algorithm

4 Self-Configuring RED

From the discussion in the previous sections, it is clear that adapting RED parameters based on traffic load and using explicit congestion notification can be beneficial to network performance. Figure 8 presents an on-line algorithm for adaptively changing the RED parameters according to the observed traffic. The idea behind this algorithm is to infer whether 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 too aggressive. On the other hand, if it oscillates around max_{th}, then the early detection mechanism is too conservative. Based on the observed queue length dynamics, the algorithm adjusts the value of max_p . In particular, it scales max_p by constant factors of α and β depending on which threshold it crosses.

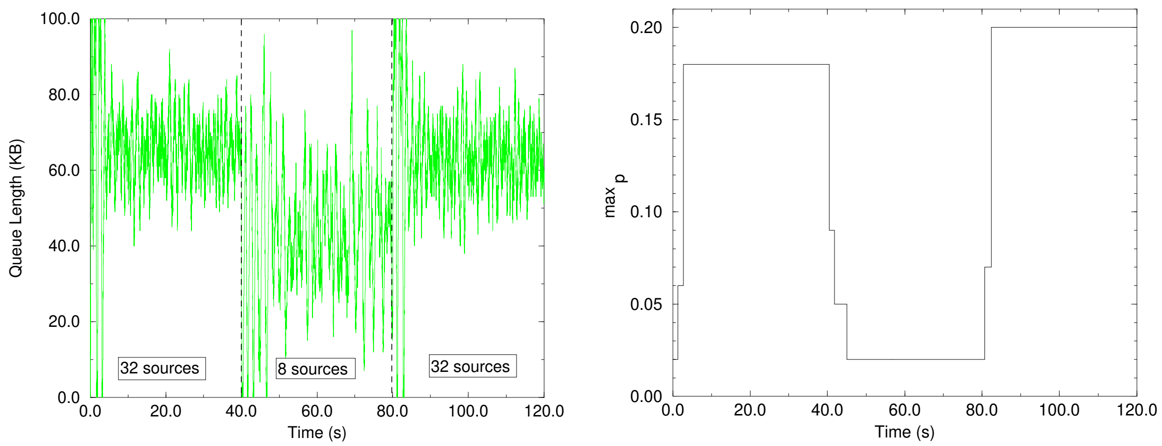
To explore the feasibility of the approach proposed in Figure 8, we ran another set of experiments using the same network as shown in Figure 1 with $100KB$ queues. In this experiment, we vary the number of active connections between 8 and 32 over 40 second intervals. Figure 9 plots the queue-lengths for RED queues statically configured to be either aggressive or conservative. When aggressive early detection is used, as shown in Figure 9(a), the RED queue performs well whenever 32 connections are active. When only 8 connections are active though, 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 9(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 10(a) shows the queue-length plot of the same experiment using the adaptive RED algorithm with α and β set to 3 and 2, respectively. We initially set max_p to 0.020 and allow the algorithm to adjust the parameter according to Figure 8. 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 10(b) plots the max_p parameter as the RED queue adapts itself to the input traffic. As expected, its value adapts to reflect



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

Figure 9: Static random early detection



(a) Queue Length (b) max_p parameter

Figure 10: Adaptive random early detection

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.

5 Implementation

We have implemented the adaptive RED modifications on a small testbed of PCs (Figure 12) running FreeBSD 2.2.6 and ALTQ [2]. In our implementation, calls to the generic `IF_ENQUEUE` and `IF_DEQUEUE` macros from the `if_output` and `if_start` are changed in order to replace the FIFO queueing mechanism typically used in BSD UNIX with the adaptive RED queueing discipline. We have also implemented ECN using

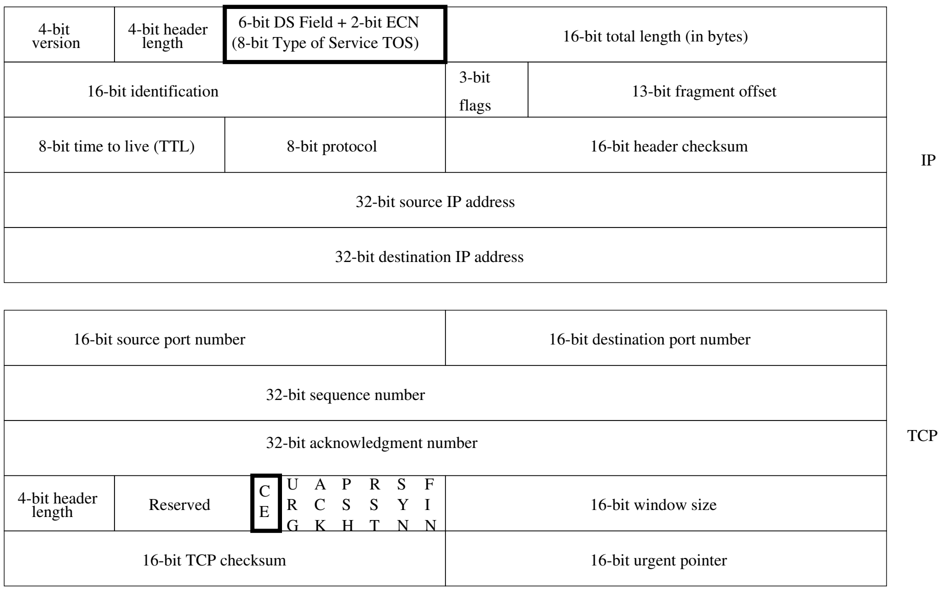


Figure 11: TCP/IP Headers

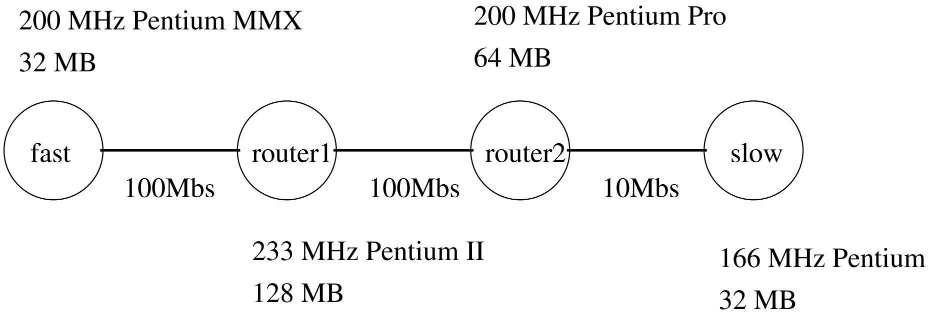
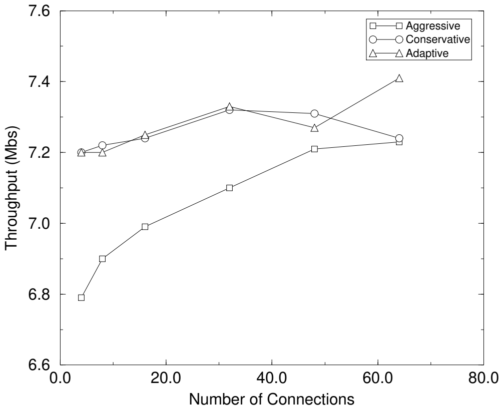


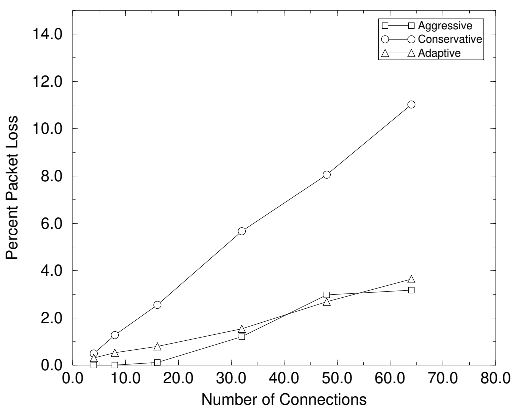
Figure 12: Testbed

two bits of the type-of-service (ToS) field in the IP header as shown in IP header part of Figure 11. When the adaptive RED algorithm decides that a packet must be dropped or marked, it examines one of the two bits to see if the flow is ECN-capable. If it is not ECN-capable, the packet is simply dropped. If the flow is ECN-capable, the other bit is set and used as a signal to the TCP receiver that congestion has occurred. The TCP receiver, upon receiving this signal, modifies the TCP header of the return acknowledgment using a currently unused bit in the TCP flags field. As Figure 11 shows, a bit labeled "congestion experienced" (CE) is used by the TCP receiver to indicate the presence of congestion to the sender. Upon receipt of a TCP segment with CE bit set, the TCP sender invokes congestion control mechanisms as if it had detected a packet loss.

Using this implementation, we ran several experiments on the testbed shown in Figure 12. Each network node and link is labeled with CPU model and link bandwidth, respectively. Note that all links are shared ethernet segments. Hence, the acknowledgments on the reverse path collide and interfere with data packets



(a) Throughput



(b) Percent Packet Loss

Figure 13: Queue Management Performance

on the forward path. As the figure shows, FreeBSD-based routers using the adaptive RED queue management algorithm connect ethernet and fast ethernet segments.

We use `netperf` [19] to generate load on the system. `netperf` is a well-known, parameterizable tool for generating network load in order to evaluate the performance of both end-hosts and network elements. In the experiments shown here, a variable number of infinite `netperf` sessions is run from `fast` to `slow`, the endpoints of the network. The router queue on the congested interface on `router1` to the slow ethernet segment is sized at $50KB$ using a min_{th} of $10KB$ a max_{th} of $40KB$. A max_p value of 0.02 is used for the conservative early detection algorithm while a max_p value of 1.00 is used for the aggressive early detection algorithm. The max_p value of the adaptive algorithm is initially set at 0.02 and allowed to vary according to the algorithm in Figure 8. In order to ensure the adaptive RED modifications did not create a bottleneck in the router, we ran several experiments between `fast` and `router2` using the modifications on `router1` to forward packets between both hosts. In all of the experiments, the sustained throughput of `router1` was always above $70Mbs$. Thus, experiments run from `fast` to `slow` always bottleneck at the output interface to the slow ethernet segment on `router1`.

Figure 13 shows the throughput and packet loss rates at the bottleneck link across a range of workloads. The throughput measures the rate at which packets are forwarded through the congested interface. This rate slightly overestimates the end-to-end goodput of the `netperf` sessions since retransmissions are counted. The packet loss rate measures the ratio of the number of packets dropped at the queue and the total number of packets received at the queue. In each experiment, throughput and packet loss rates were measured over ten 5-second intervals and then averaged. As Figure 13(a) shows, both the conservative and adaptive early detection algorithms maintain high throughput levels across all workloads while the aggressive early detection algorithm achieves a lower throughput for smaller numbers of connections. Note that since the ethernet segments are shared, acknowledgments on the reverse path collide with data packets on the forward path, thus limiting throughput. Figure 13(b) shows the packet loss rates over the same workloads. As the figure shows, both the aggressive and the adaptive early detection algorithms maintain low packet loss rates across all workloads while the conservative early detection algorithm suffers from fairly large packet loss rates as the number of connections increase. As the number of connections increases, an interesting observation is that the marking rate of the adaptive and aggressive early detection algorithms approaches 50%. Because aggregate TCP behavior becomes more aggressive as the number of connections increases, it becomes more and more difficult for the RED queue to maintain low loss rates. Fluctuations in queue lengths occur so abruptly that the RED algorithm oscillates between periods of sustained marking and packet loss and periods of minimal marking and link underutilization.

6 Conclusion and Future Work

This paper has shown how adaptive active queue management can be used in conjunction with explicit congestion notification to effectively reduce packet loss in congested networks. In particular, an adaptive RED mechanism which is cognizant of the number of active connections can provide significant benefits in terms of decreasing packet loss and increasing network utilization. Preliminary experiments using a small testbed has shown the efficacy of the algorithm. Since early detection algorithms become more effective as round trip times increase, it is expected that the performance improvements will become even more significant as the network grows.

We are currently examining ways to methodically improve 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. Another mechanism would be to infer the number of connections present by the rate that the average queue length changes, then have the RED queue adapt its parameters accordingly. It may also be possible to adapt other RED parameters such as inter-packet drop probabilities, RED threshold values, or queue length weights in order to optimize performance. For example, one could adapt the weight used to calculate the exponentially weighted average of the queue length based on the input traffic. When a large number of connections are present, queue lengths can fluctuate considerably in a short period of time. In such cases, an average which heavily weights more recent queue lengths is necessary in order to trigger congestion notification in time to prevent packet loss.

We are also examining ways of applying the results of these experiments to other areas such as differentiated services and end-host congestion control. In particular, we plan on applying the adaptive RED techniques to an enhanced ToS-enabled version of RED (ERED) [4, 5] to improve performance. Incorporating adaptive RED and ECN into ERED can potentially alleviate many of the problems in using TCP over differentiated services networks. We are also experimenting with modifications to TCP's linear increase mechanism, in order to alleviate the quadratic dependence of congestion notification on the number of connections [3]. Without such modifications, a large percentage of packets must be marked in order to prevent packet loss. Finally, we are experimenting with a new active queue management algorithm in which marking is not triggered by either the instantaneous or average queue lengths. One of the problems with RED is that its behavior changes depending on the value of the average queue length and the queue's threshold values. This can make it difficult to predict the performance from the queue since the algorithm relies on a very subtle interplay of parameters. An active queue management algorithm which is based on a single marking probability which adapts slowly based on packet loss and link utilization may lead to smoother marking and more predictable performance.

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] K. Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. *USENIX 1998 Annual Technical Conference*, June 1998.

[3] W. Feng, D. Kandlur, D. Saha, and K. Shin. Techniques for Eliminating Packet Loss in Congested TCP/IP Networks. *UM-CSE-TR-349-97*, November 1997.

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

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin. Adaptive Packet Marking for Providing Differentiated Services in the Internet. *Proc. IEEE International Conference on Network Protocols*, October 1998.

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

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

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

[9] E. Hashem. Analysis of Random Drop for Gateway Congestion Control. *MIT Technical Report*, 1990.

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

[11] V. Jacobson. Presentations to the IETF Performance and Congestion Control Working Group. August 1989.

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

[13] 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.

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

[15] 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.

[16] A. Mankin. Random Drop Congestion Control. In *Proceedings ACM SIGCOMM (Special Issue Computer Communication Review)*, pages 1–7, September 1990. Published as Proc. ACM SIGCOMM '90; (Special Issue Computer Communication Review), volume 20, number 4.

[17] A. Mankin and K. K. Ramakrishnan. Gateway Congestion Control Survey. *RFC 1254*, August 1991.

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

[19] Netperf. The Public Netperf Homepage: http://www.netperf.org/. The Public Netperf Homepage, 1998.

[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] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *Proc. of ACM SIGCOMM*, pages 79–88, August 1994.

[25] B. Suter, T. V. Lakshman, D. Stiliadis, and A. Choudhury. Design Considerations for Supporting TCP with Per-flow Queueing. *Proc. IEEE INFOCOM*, March 1998.

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