IMPROVING INTERNET CONGESTION CONTROL AND QUEUE MANAGEMENT ALGORITHMS

by

Wu-chang Feng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1999

Doctoral Committee:

Professor Kang G. Shin, Chair
Assistant Professor Peter Chen
Associate Professor Farnam Jahanian
Assistant Professor Sugih Jamin
Dilip Kandlur, Manager, IBM Research
Assistant Professor Kimberly Wasserman

© Wu-chang Feng 1999
All Rights Reserved

For my parents

ACKNOWLEDGEMENTS

Throughout my graduate school career, I've been fortunate enough to have the help and support of a large number of people. I would especially like to thank

- Professor Kang Shin, my advisor, for giving me a chance to "walk on" to his team of researchers, for his patience in allowing me to explore different avenues of research, and for his guidance throughout the years.
- Dilip Kandlur and Debanjan Saha for giving me the invaluable opportunity to work at IBM Research. Thanks also to the IBM Corporation for their generosity and financial support over the last two years.
- Professors Peter Chen, Farnam Jahanian, Sugih Jamin, and Kimberly Wasserman, my committee members, for their insightful comments on the dissertation. Special thanks to Farnam for his technical and academic guidance and to Sugih for pushing me to go the extra mile in finishing the thesis.
- The National Science Foundation, the Office of Naval Research, and the Defense Advanced Research Projects Agency, for their initial financial support.
- Jim Dolter, Jennifer Rexford, and Ashish Mehra, for all of their technical and academic help and for teaching me the arcane ways of being a graduate student in computer science.
- Scott Dawson, Todd Mitton, Anees Shaikh, Tarek Abdelzaher, Rob Malan, and Steve Vlaovic, for their many useful and insightful discussions about networking and other random topics.
- B.J. Monaghan for making everything around RTCL run so smoothly and for the many years of betting on the PSU/MSU football games. Blow-pops never tasted so good.
- YF, for all of the help and support throughout the year. Food never tasted so good.
- 930 Dewey (the house that ultimate built) and MagnUM, for a great run at Nationals and for always reminding me that there's more to life than work.

Finally, I would most like to thank my parents, Tse-yun and Elaine, and my brothers, Wu-chun, Wu-chi, and Wu-che for their continual encouragement, guidance and support. This thesis is a testament to our many years of academic perseverance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

The success of the Internet can largely be attributed to the strength of its protocols. By providing users and developers with robust, interoperable services, the Internet has effectively provided all of the essential building blocks for constructing applications such as the WWW. As one looks at almost all of the applications in widespread deployment on the Internet today, it is no coincidence that they are all built upon TCP/IP. Over the last decade, TCP/IP has consistently met the challenge of new applications and has been able to provide a solid foundation from which to build them.

With the rapid proliferation of the WWW, the Internet has seen an enormous growth in both the demand for access from its users and in the demand for new services from its applications. As a result of these new challenges, weaknesses in TCP/IP have become increasingly apparent. Rising packet loss rates and decreasing network efficiency have caused significant problems to users. In addition, the inability to support new services has severely hindered the widespread deployment of bandwidth-sensitive applications. This thesis focuses on these extremely important challenges to today's Internet and describes how current congestion control and queue management techniques can be modified to solve them.

1.1 Congestion Control in the Internet

At the heart of TCP/IP's success over the last decade is its ability to deliver service in times of extremely high demand. The key reason behind this is TCP's congestion control mechanisms [34]. The idea behind TCP congestion control is to control network load by having sources adjust their rates according to the level of congestion in the network. More specifically, if a TCP source detects or observes packet loss based on information on the packets which it has sent, it backs off its sending rate in order to avoid further packet loss and congestion. If a TCP source observes that all of its packets are being delivered, it slowly increases its sending rate in order to fully utilize network capacity. In this way, TCP has been able to effectively minimize packet loss while maximizing network utilization over the last decade.

Recently, as demand for access has outpaced the ability for providers to upgrade network paths, the ability of TCP to provide best-effort service efficiently has deteriorated. In particular, an alarming rise in packet loss rates has been observed across a number of network links [53]. This rise in packet loss has resulted in the steady decline of network efficiency as sources and routers continually generate and forward packets which are then dropped. In order to address the steady rise in packet loss rates in the Internet, the Internet Engineering Task Force (IETF) is considering the deployment of Explicit Congestion Notification (ECN) [23, 55] and active queue management [4, 26] as a means to prevent packet loss. The idea behind ECN is to give the network the ability to explicitly signal TCP sources of congestion and to have the TCP sources reduce their transmission rates in response to the signal. Since TCP sources currently only reduce their transmission rates upon detecting a packet loss, without ECN, the amount of packet loss observed across the Internet will always remain non-zero. While ECN provides the network a mechanism for reducing packet loss, it must be used in conjunction with active queue management in order to be effective. The goal of active queue management is to detect congestion early and to convey congestion notification to sources before queue overflow and packet loss occur. By decoupling congestion notification from packet loss and using active queue management mechanisms, it is the hope of the IETF that packet loss rates in the Internet can be controlled.

1.2 Quality of Service in the Internet

The service realized by TCP over today's Internet is commonly known as "best-effort". Using simple FIFO queueing in the network combined with TCP congestion control at the end points, sources maintain approximate fairness between themselves when they are multiplexed over the same bottleneck link. As the need for new services has grown, the lack of service differentiation in the network has become problematic. As a result, a growing number of applications such as multimedia streaming applications continually circumvent TCP and its congestion control mechanisms in favor of UDP and their own rate control mechanisms.

In an attempt to address the growing needs of applications, the IETF has developed a number of architectural enhancements to the current Internet infrastructure which allow the network to provide predictable services to applications on an end-to-end basis. The result of this effort has been the standardization of the Resource Reservation Setup Protocol (RSVP) [6, 69] and its associated suite of service classes [59, 68]. In this approach, individual applications signal their resource requirements to the network on an end-to-end basis. Given this, intermediate network elements (routers, switches, etc.) set aside the appropriate amount of resources for the application. When subsequent packets arrive at each network element, they are then scheduled in a manner which satisfies the requirements of the application. While this service architecture provides a solid foundation for providing different classes of service in the Internet, it mandates fairly significant changes to the network. In addition, support for such services can add a significant amount of overhead in packet processing within the network. Because of this, the IETF is also considering a more evolutionary approach to provide service differentiation in the Internet. This approach, as outlined by the Differentiated Services (DIFFSERV) working group, relies on the use of the type-of-service bits (ToS) bits in the IP header [2, 13, 54, 58] to provide coarse-grained quality of service to applications. The goal of the DIFFSERV effort is to define a minimal set of building blocks which can be used to construct a variety of services to emerging applications.

1.3 Structure of Thesis

This thesis presents effective techniques for supporting an explosion in the number of users and for supporting a myriad of new applications which require more out of the network than the best-effort service the current Internet infrastructure affords. Chapter 2 surveys related work on controlling congestion and on providing quality of service in the Internet. Chapter 3 and Chapter 4 address the problem of maximizing network efficiency in times of extremely heavy congestion. Chapter 3 demonstrates a significant weakness in current active queue management techniques in that they are not sensitive to the level of congestion in the network. In order to address this shortcoming, an adaptive queue management algorithm which can effectively reduce packet loss over a wide range of workloads is developed, implemented and evaluated. While adaptive queue management provides some benefit, Chapter 3 also shows that high packet loss rates are, in part, an artifact of TCP congestion control. As a result, a conservative modification to TCP's congestion control algorithm is proposed and evaluated. Together, these two modifications can provide an *order of magnitude* improvement in packet loss rates using a significantly smaller amount of buffer space in the network. Chapter 4 extends this work by addressing several problems with current active queue management algorithms. In particular, this chapter demonstrates that the dependence on queue lengths to perform congestion management is inherently flawed. To address this limitation, BLUE, a fundamentally different active queue management algorithm is proposed, implemented, and evaluated. This algorithm outperforms all current active queue management algorithms by a large margin in terms of packet loss rates and buffer space requirements. In addition, this chapter also proposes an extension to BLUE which allows the network to scalably enforce fairness between a large number connections. This extension uses an extremely small amount of buffer space and state.

Chapter 5 and Chapter 6 address the problem of supporting quality of service across the Internet. In particular, these chapters focus on building scalable, deployable mechanisms for supporting bandwidth guarantees across the Internet based on the DIFFSERV approach. Chapter 5 first demonstrates how DIFFSERV-style mechanisms can be ineffective in providing predictable service to applications. One of the major problems is that the end-to-end rate control mechanism of TCP is disjunct from the rate-based marking being done in the network. In order to address this problem, this chapter proposes and evaluates several modifications to the congestion control mechanisms at the end host in order to fully take advantage of DIFFSERV-marking in the network and to deliver predictable service to applications. Chapter 6 extends this work by developing an architecture for providing soft bandwidth guarantees in a scalable, easily deployable, manner. In particular, the the bandwidth sharing and marking behavior of connections using DIFFSERV-style marking is analyzed and some of the weaknesses of current approaches are shown. This chapter then describes novel mechanisms for integrating packet marking into end hosts in order to (1) obtain optimal marking rates between sources, (2) detect heterogeneity and lack of service differentiation in the network, and (3) allow for incremental deployment. Finally, Chapter 7 concludes with a summary of research contributions.

CHAPTER 2

BACKGROUND

This chapter reviews the important features of today's congestion control and queue management algorithms as well as the current state of providing quality of service in the Internet.

2.1 TCP and Active Queue Management

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 are wasted. In extreme cases, this situation can lead to congestion collapse [34]. Loss rates are especially high during times of heavy congestion, when a large number of connections compete for scarce network bandwidth. With the explosion of the WWW, recent measurements have shown that the growing demand for network bandwidth has driven loss rates up across a number of congested links in the Internet [53].

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 (CWND) 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 rate at which TCP sends its packets eventually exceeds the network's capacity to deliver them. When this happens, queues build up in the network routers and overflow, causing packets to be dropped. TCP assumes that all packet loss is due to congestion and reduces its congestion window upon detecting a loss. TCP's congestion control algorithm is fairly straightforward. When a connection starts up, it attempts to ramp up its sending rate quickly by exponentially increasing its congestion window until it reaches an implementation-specific value (SSTHRESH). This stage is called *slow-start* and allows the source to double its congestion window, and thus its sending rate, every round-trip time. In order to prevent excessive losses due to an exponentially-increasing sending rate, TCP senders typically employ what is known as the congestion-avoidance algorithm [34, 61], a modification to TCP first deployed in Reno variants of TCP. In this algorithm, TCP uses the SSTHRESH value to approximate 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 offered load increases above network capacity, packets are eventually dropped. One way in which TCP detects a packet loss is through the receipt of a number of duplicate cumulative acknowledgments from the receiver [35]. Upon receiving a given number of duplicate acknowledgments, TCP infers that a packet loss has occurred and immediately reduces its sending rate in half by halving its congestion window and sets SSTHRESH to the new value of the 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 separate, retransmission timeout mechanism to trigger subsequent retransmissions of lost packets. When a retransmission timeout occurs, 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 the 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 sending.

Figure 2.1 shows a graphical picture of how TCP slow-start and congestion avoidance work. As the figure shows, TCP initially starts with a congestion window of 1. The window is then doubled every round-trip time. When the congestion window reaches SSTHRESH, TCP slows its rate of increase. Eventually, when the transmission rate of the connection overwhelms the bottleneck link, packets are dropped. This loss is detected by TCP which then reacts by halving the congestion window (assuming the fast-retransmit and fast-recovery mechanisms are triggered). As the figure shows, upon recovering from congestion, the TCP sender enters the congestion avoidance phase

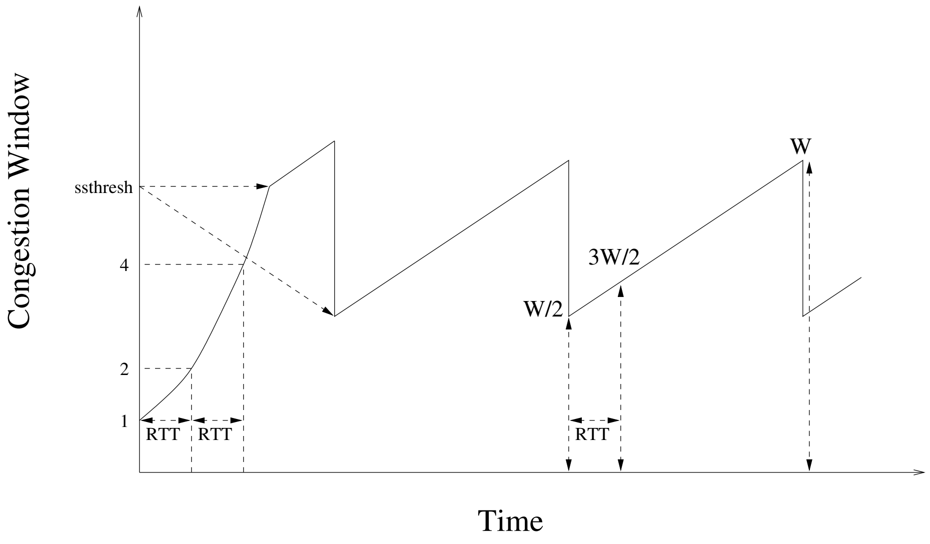


Figure 2.1: Example of TCP congestion window behavior

in which the window is increased linearly at a rate of one segment per round trip time. In steady state, TCP then oscillates between a window of W and $\frac{W}{2}$ where W depends on the capacity of the network and the number of connections currently active over the bottleneck link.

Given the importance of TCP and its congestion control mechanisms to the health of the Internet, there have been a number of proposed modifications to its algorithms. One modification which has been proposed is selective acknowledgments (SACK) [45]. SACK augments TCP's cumulative acknowledgment mechanism with additional information that allows the receiver to inform the sender which segments it is missing. By specifying this information, the TCP sender can make more intelligent decisions in determining when packets have been lost and in identifying which segments should be retransmitted. This helps TCP detect congestive loss more quickly and eliminates unnecessary retransmissions by TCP senders. Another set of proposed TCP modifications focuses on congestion recovery. TCP is ACK-clocked, often sending only after it has received acknowledgments for previously transmitted packets. When there are insufficient packets or acknowledgments in flight to trigger TCP sends, a retransmission timeout must occur before the TCP source can resume sending. Because the Reno variant of TCP freezes its window while recovering from congestion, it often induces a subsequent retransmission timeout since the source does not send packets upon receiving acknowledgments in the recovery phase. To address this problem, a simple observation is made. When a TCP sender receives any type of acknowledgment, it is a signal that a packet has

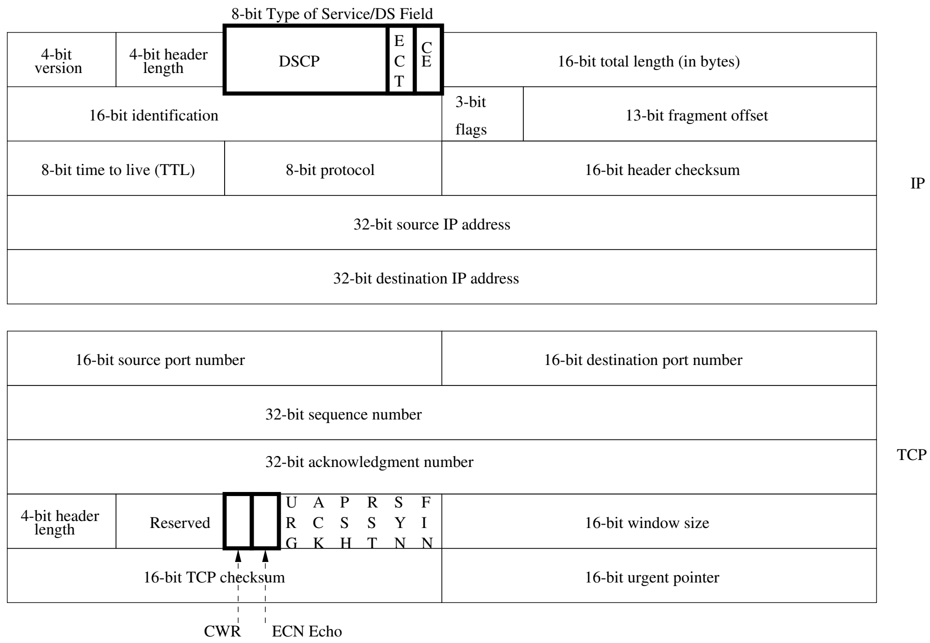


Figure 2.2: TCP/IP headers

left the network and should thus allow the TCP sender to inject an additional packet without causing further congestion. This modification allows TCP to maintain its ACK-clocking and prevents unnecessary retransmission timeouts. Both the FACK [44] and NewReno [24, 32] modifications use this observation to improve TCP performance. Finally, more radical changes to TCP's congestion control algorithms have been proposed. In current incarnations of TCP, the congestion window follows a sawtooth-like pattern where the congestion window is continually increased until packet loss occurs. While this allows TCP to probe for additional bandwidth, such behavior eventually induces packet loss. The idea behind the Tri-S [66, 67] and Vegas [7] modifications is to change the congestion avoidance phase so that it only performs its linear increase when the network is not congested. In both algorithms, if the round-trip times indicate an increase in delay due to queues being built up in the network, the TCP source either decreases or fixes the size of the congestion window rather than increasing it. While these mechanisms have the potential for improving loss rates in the Internet, it is unclear how well each scheme performs when congestion is persistent. In addition, by modifying the linear-increase/multiplicative-decrease algorithm of TCP, these modifications cannot ensure that max-min fair sharing occurs between connections which are multiplexed across the link [8, 38].

With the exception of Tri-S and Vegas, one of the problems with the TCP congestion control algorithm over current networks is that the sending sources reduce their transmission rates only

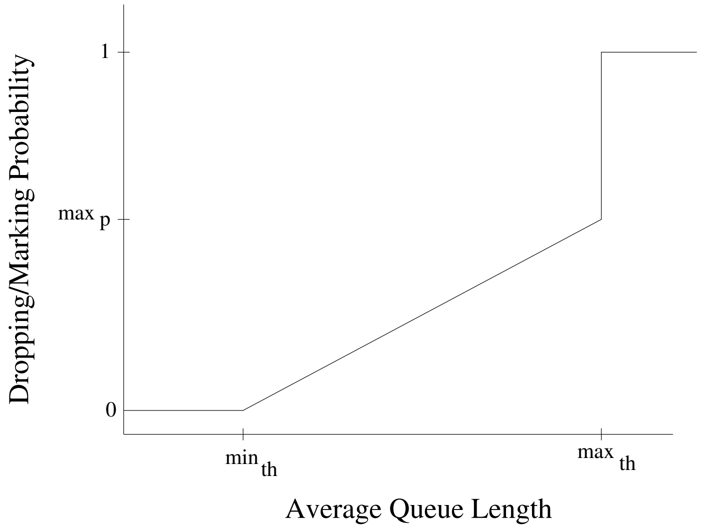


Figure 2.3: The marking/dropping behavior of RED

after detecting packet loss due to queue overflow. This is a problem since a considerable amount of time may pass between when the packet is dropped at the router and when the source actually detects the loss. In the meantime, a large number of packets may be dropped as sources continue to transmit at a rate that the network cannot support. Because of this, the IETF is advocating the use of explicit congestion notification (ECN) [23, 55] and active queue management as a means to prevent packet loss. The idea behind ECN is to decouple packet loss from congestion notification. In this proposal, ECN is implemented using two bits of the type-of-service/DS field of the IP header and two bits of the currently reserved flags field of the TCP header as shown in Figure 2.2. When a network router experiences congestion, it can explicitly signal the sources instead of dropping their packets. In order to do so, the router first examines the ECN-capable Transport bit (ECT) 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 congestion experienced bit (CE) 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 2.2 shows, a bit labeled "ECN-echo" is used by the TCP receiver to indicate the presence of congestion to the sender. Upon receipt of a TCP segment with the ECN-echo bit set, the TCP sender invokes congestion control mechanisms as if it had detected a packet loss. In addition, it sets the "Congestion Window Reduced" (CWR) bit of its next packet to the receiver in order to signal the receiver that it has, in fact, reduced its sending rate.

In conjunction with ECN, the IETF is also advocating the use of active queue management. The idea behind active queue management is to detect incipient congestion *early* and to convey congestion notification to the end hosts, in order to allow them to reduce their transmission rates 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) [4, 26]. RED maintains an exponentially weighted moving average (EWMA) of the queue length which it uses to detect congestion. RED detects increases in the average queue length and uses it to determine whether or not to drop or ECN-mark a packet. More specifically, Figure 2.3 plots the marking/dropping probability of RED as a function of the average queue length. As the figure shows, when the average queue length exceeds a minimum threshold (min_{th}), packets are randomly dropped or marked 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 and an initial probability parameter (max_p). As Figure 2.3 shows, the calculated marking/dropping probability is a linear function of the average queue length. The probability is 0 when the average queue length is less than or equal to min_{th} and linearly increases to max_p when the average queue length approaches a maximum threshold (max_{th}). When the average queue length exceeds max_{th}, all packets are dropped or marked.

With the deployment of ECN and RED, it is the hope of the IETF that packet loss rates in the Internet can be controlled. Unfortunately, as Chapter 3 and Chapter 4 show, there are significant weaknesses in both TCP congestion control and in RED queue management which prevent packet loss from being eliminated. Given these weaknesses, a number of congestion control and queue management algorithms which effectively prevent packet loss are proposed and evaluated.

2.2 Integrated and Differentiated Services

As the Internet evolves, the number of diverse applications being deployed has increased significantly. Unfortunately, many of these applications require more stringent performance guarantees in terms of bandwidth and end-to-end delay than the current Internet infrastructure and its best-effort service provide [5, 12]. Because the best-effort service model in place today cannot support every

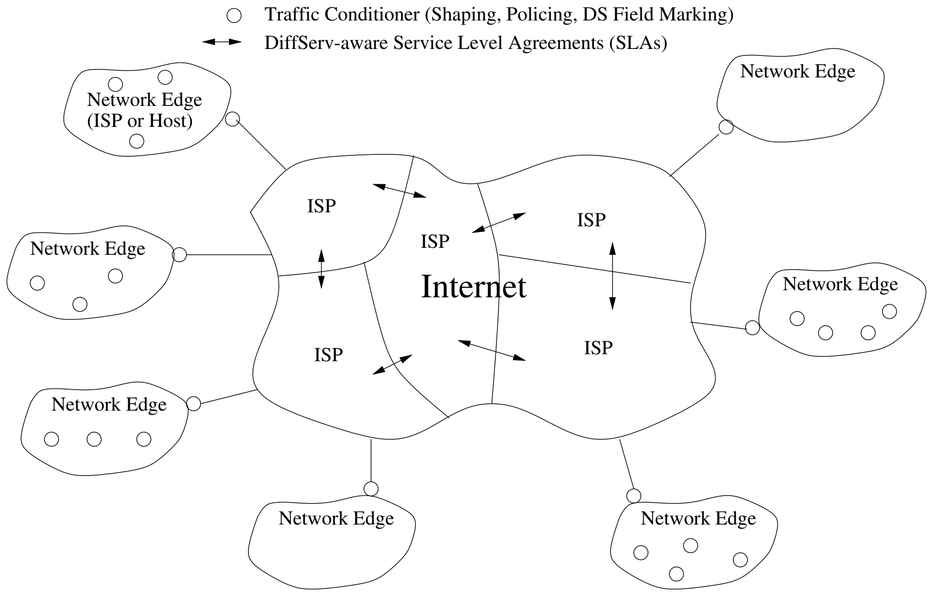


Figure 2.4: DIFFSERV architecture

application, a great deal of effort has been taken to construct additional services in order to meet the demands of new applications.

In an attempt to enrich and augment the services provided by the network, the Internet Engineering Task Force (IETF) has developed a number of architectural extensions that permit the allocation of different levels of service to different users. One of the outcomes of this effort is an architecture that provides service discrimination by explicit allocation and scheduling of resources in the network. This model, based on the Resource Reservation Setup Protocol (RSVP) [6, 69] and its associated suite of service classes [59, 68], is the Internet incarnation of the traditional "circuit-based" quality of service architecture. While this service architecture provides a solid foundation for providing different classes of service in the Internet, it mandates significant changes to the Internet infrastructure. Because of this, a more evolutionary approach to provide service differentiation in the Internet using the type-of-service (ToS) bits in the IP header [2, 13, 54, 58] has recently gained a lot of momentum. Through the Differentiated Services (DIFFSERV) working group, these bits, as shown in Figure 2.2, have been renamed as the "DS field" and the functions associated with them have been redefined. The crux of the DIFFSERV approach is to standardize a simple set of mechanisms for handling packets with different priorities [3, 11, 19, 50], as encoded in the DS field of the IP header. Figure 2.4 shows the basic architecture of the DIFFSERV approach. As the figure shows, traffic conditioners such as shapers, DS-markers, and droppers are placed at the edges of the network. Given this functionality at the edge, the interior routers then use the priority markings of packets in order to deliver differentiated services to various packets. This provides a very basic QoS architecture in which most of the complexity is pushed to the edges of the network where it is the most scalable.

Because of the limited amount of bits available for use in the DS field, the DIFFSERV working group has defined a small set of building blocks which are used by routers to deliver a number of services. These building blocks, called per-hop behaviors (PHBs), are encoded in the Differentiated Services Codepoint (DSCP) part of the DS field and specify the forwarding behavior each packet receives by individual routers in the Internet. When used on an end-to-end basis, it is envisioned that these building blocks can be used to construct a variety of services which are able to support a range of emerging applications. Among the initial PHBs being standardized are the Expedited Forwarding (EF) [37] and the Assured Forwarding (AF) [31] PHBs. The EF PHB specifies a forwarding behavior in which packets see a very small amount of loss and a very low queueing delay. In order to ensure every packet marked with EF receives this service, EF requires every router to allocate enough forwarding resources so that the rate of incoming EF packets is always less than or equal to the rate at which the router can forward them. In order to preserve this property on an end-to-end basis, EF requires that traffic be shaped and reshaped in the network. The AF PHB group, on the other hand, specifies a forwarding behavior in which packets see a very small amount of loss. The AF PHB group consists of four, independently forwarded classes. Within each class, two or three drop preference levels are used to differentiate between flows in the class. The idea behind AF is to preferentially drop best-effort packets and packets which are outside of their contract when congestion occurs. By limiting the amount of AF traffic in the network and by managing the best-effort traffic appropriately, routers can then ensure low loss behavior to packets marked with the AF PHB.

While it is relatively clear how to build predictable services using the protocols and mechanisms provided by RSVP and INTSERV, the ability to construct predictable services using the coarse-grained mechanisms provided by DIFFSERV is an open issue. Since DIFFSERV specifies only the local forwarding behavior given to packets at individual routers, one of the biggest challenges is to be able to concatenate DIFFSERV mechanisms on an end-to-end basis to construct useful services for applications. Chapter 5 and Chapter 6 address some of the problems in providing predictable end-to-end services using DIFFSERV mechanisms based on the AF PHB. Key among these problems is the complex interaction of current congestion control and queue management algorithms with the priority marking and handling of packets in the network. As a result of these problems, an architecture and a number of mechanisms are described which allow applications to effectively take advantage of DIFFSERV support.

CHAPTER 3

TECHNIQUES FOR ELIMINATING PACKET LOSS IN CONGESTED TCP/IP NETWORKS

3.1 Introduction

As described in Chapter 2, one of the reasons for high packet loss rates is the failure of the network to provide early congestion notification to sources. This has led to proposals for active queue management such as RED and variations thereof [26, 41]. While RED certainly outperforms traditional drop-tail queues, this chapter shows 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 which is high enough to prevent packet loss due to buffer overflow, while low enough to prevent underutilization of the bottleneck link.

This chapter demonstrates the ineffectiveness of the current RED queue management algorithm and shows how RED queues can be self-parameterized 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. 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. This chapter also 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 the linear increase algorithm 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.

3.2 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 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 $(1 - \frac{1}{2N})$. As N becomes large, the impact of individual congestion notifications decreases. Without modifying the RED algorithm 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, without modifying the RED algorithm 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 on-line mechanisms for optimizing performance.

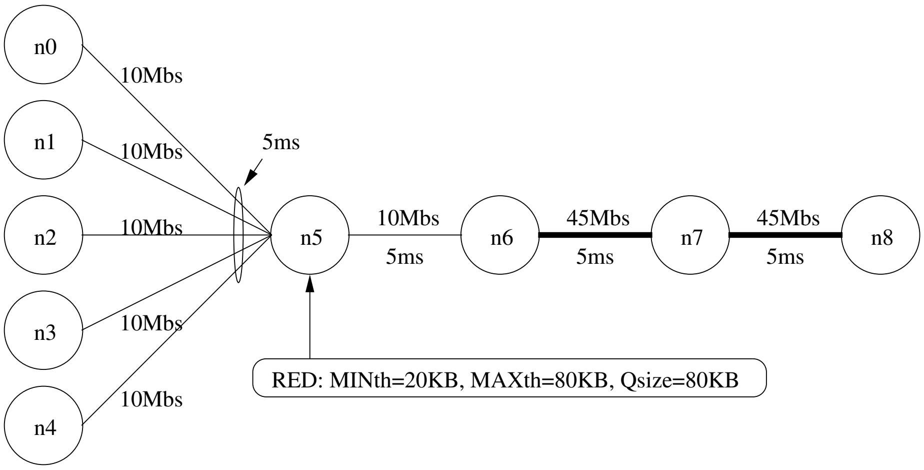
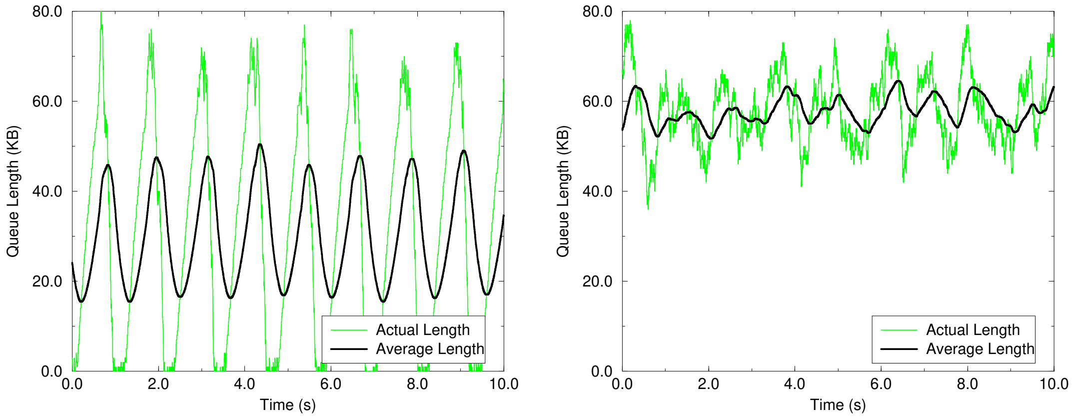


Figure 3.1: Network topology

3.2.1 Traffic load and early detection

To examine the impact that traffic load has on early detection mechanisms, a set of experiments using the `ns` simulator [46] was performed. The `ns` simulator has been used extensively in a number of studies reported in the literature. While `ns` does not use production TCP code, it implements congestion and error control algorithms used in different implementations of TCP with remarkable accuracy. In these experiments, both the aggressiveness of the early detection algorithm and the total number of connections multiplexed on the bottleneck link were varied. Figure 3.1 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. The performance of of RED using ECN [23] is examined first. By using RED and end host TCP sources enabled with ECN, 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}, the max_{th} parameter is set 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 size are described in Section 3.2.2. Figure 3.2 shows the queue length plot of the congested queue located from $n5$ to $n6$ when there are 8 and 32 connections simultaneously competing for bandwidth over the link. In these experiments, the RED algorithm is made aggressive by changing max_p, RED's initial drop probability. As Figure 3.2(a) shows, when only 8 connections are active, aggressive early detection sends congestion notification back to the



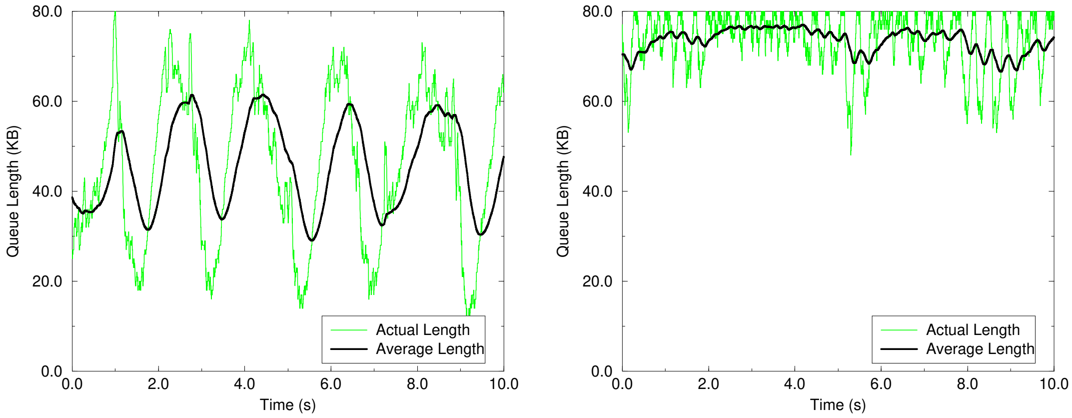
(a) 8 Connections (b) 32 Connections

Figure 3.2: Aggressive early detection (max_p = 0.250)

sending sources at a rate which is too high, causing the offered load to be significantly smaller than the bottleneck link bandwidth at certain times. This causes periodic underutilization where the queue is empty and the bottleneck link has no packets to send. Figure 3.2(b) shows the queue plot when the number of connections is increased to 32. In contrast, aggressive early detection performs as desired, sending congestion notification at a rate which can both avoid packet loss and achieve high link utilization.

Figure 3.3 shows the same set of experiments using conservative early detection. In contrast to Figure 3.2(a), Figure 3.3(a) shows that by using less aggressive early detection, the RED queue can maintain high link utilization while avoiding packet loss over smaller numbers of connections. However, when the number of connections is increased to 32, as Figure 3.3(b) shows, conservative early detection does not deliver enough congestion notification to the sending sources. Thus, the queue continually overflows causing the RED queue 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 TCP is not aggressive enough in backing off its sending rate in response to congestion and that the packets which are successfully delivered through the bottleneck queue are enough to trigger subsequent rate increases in the sending TCP sources. Thus, the bottleneck queue remains close to fully occupied through the duration of the experiment.

To quantitatively evaluate the impact of max_p, the experiments were repeated across a range of traffic loads and the loss rates and link utilizations observed were plotted. In each experiment, connections are started within the first 10 seconds of simulation. After 100 seconds, both the loss

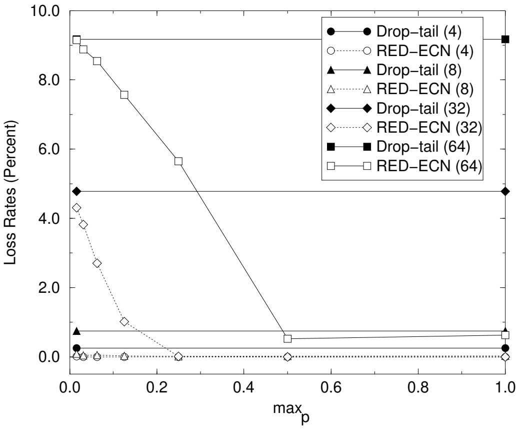


(a) 8 connections

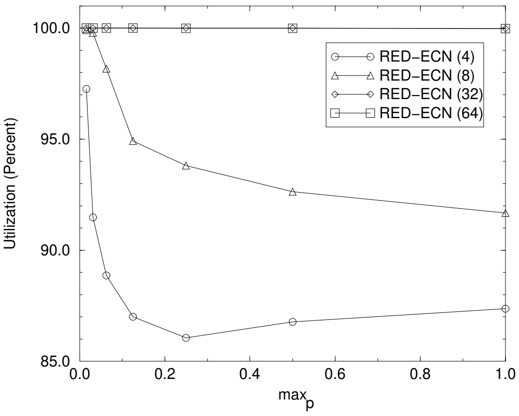
(b) 32 connections

Figure 3.3: Conservative early detection (max_p = 0.016)

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. Link utilization is calculated as the total number of packets sent divided by the maximum number of packets the link could send. Figure 3.4(a) shows the loss rates observed for experiments using 4, 8, 32, and 64 connections. The figure 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. There are two main reasons why this is the case. One reason, as described earlier, is that with a large number of connections, it takes a larger amount of congestion notification (i.e. packet drops), to sufficiently signal the end hosts to back off their sending rates. The other reason is due to a fundamental problem with TCP congestion control which is described in Section 3.3. Figure 3.4(a) also shows the loss rates using RED-ECN over a range of max_p values. The corresponding bottleneck link utilization for each experiment is shown in Figure 3.4(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 from occurring. In order to optimize performance in this case, early detection must be made aggressive.



(a) Packet loss rates

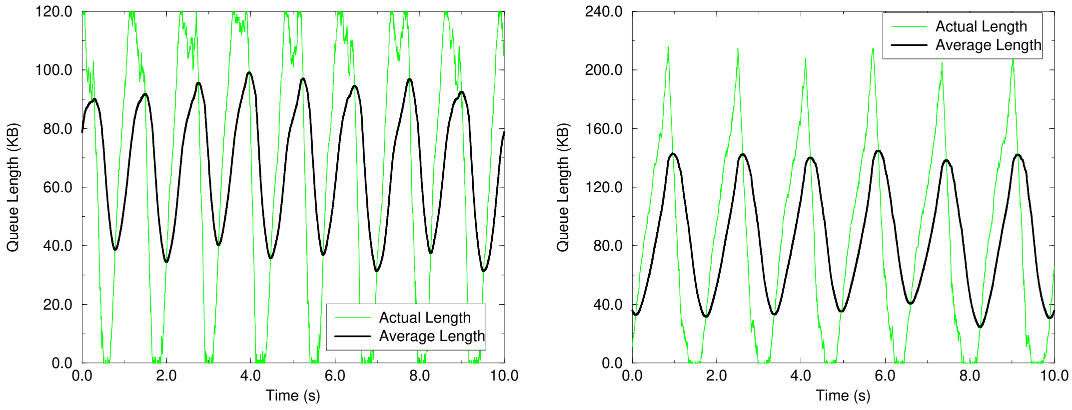


(b) Bottleneck link utilization

Figure 3.4: Impact of early detection aggressiveness on RED-ECN

3.2.2 Avoiding deterministic congestion notification

In the previous section, max_{th} is set 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 3.5 shows the queue length plot using the same experiment as in Figure 3.3(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. The plot 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 sending rates of the sources. This is in contrast to the behavior seen in Figure 3.3(b). In that experiment, a connection which was able to send a packet through the bottleneck link always increased its sending rate even though the bottleneck queue was 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 3.5(a) 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 plot shows, periods of congestion are immediately followed by fairly long periods of underutilization where the queue is empty. Furthermore, it takes a large amount of extra buffer space in order to ensure that no loss occurs. Figure 3.5(b) shows the queue length plot



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

Figure 3.5: Impact of max_{th} and queue size

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.

3.2.3 Adaptive RED

From the previous experiments, it is clear that more aggressive early detection is needed when a large number of flows are active in order to avoid packet loss and deterministic congestion notification. Similarly, less aggressive early detection is needed when a small number of flows are active in order to prevent underutilization. Because adapting RED parameters can be beneficial to network performance, this section proposes an on-line mechanism for adaptively changing the parameters according to the observed traffic. This algorithm, called Adaptive RED, is shown in Figure 3.6. The idea behind this algorithm is to infer whether or not RED should become more or less aggressive by examining the average queue length behavior. If the average queue length continually crosses back and forth over min_{th}, then the early detection mechanism is being too aggressive. If the average queue length continually crosses back and forth over max_{th}, then the early detection mechanism is not aggressive enough. Given the behavior of the average queue length, the algorithm then adjusts its value of max_p accordingly. For this algorithm, max_p is simply scaled by constant factors of α and β depending on which threshold it crosses. Figure 3.7 shows how Adaptive RED changes the marking/dropping behavior of RED. In contrast to the original RED algorithm as shown in Figure 2.3, Adaptive RED's marking function changes depending on the setting of max_p. In times of

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 = $max_p \times \beta$;

Figure 3.6: Adaptive RED algorithm

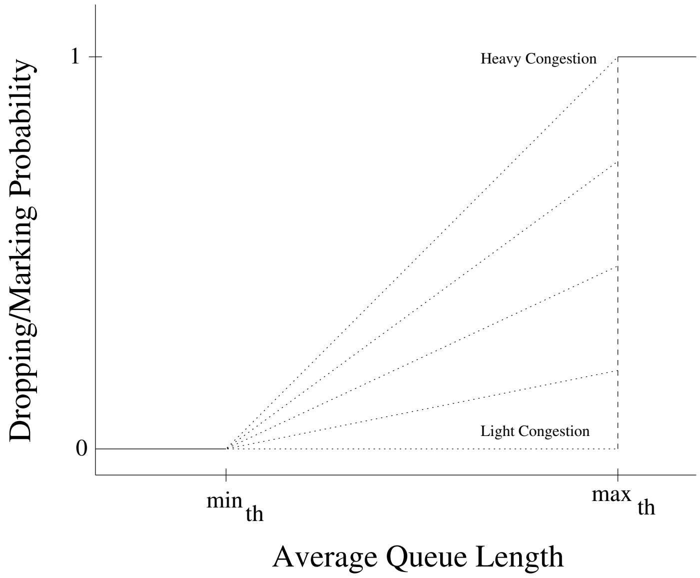
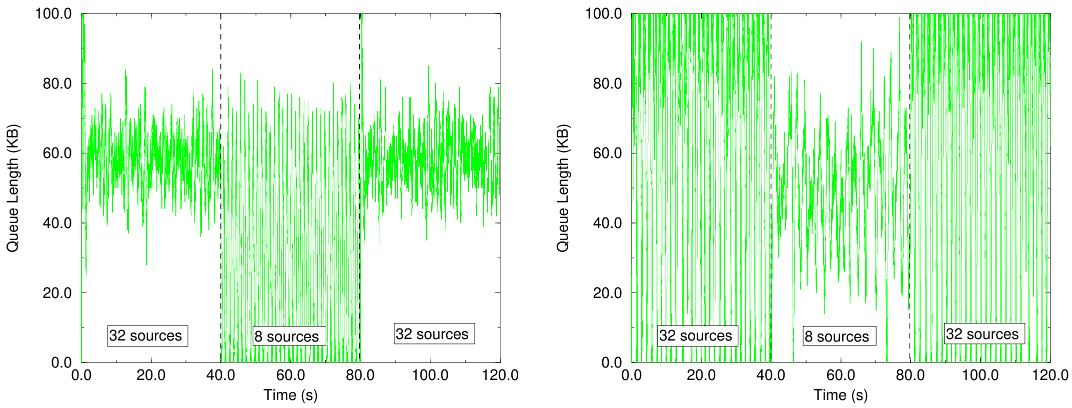


Figure 3.7: The marking/dropping behavior of Adaptive RED



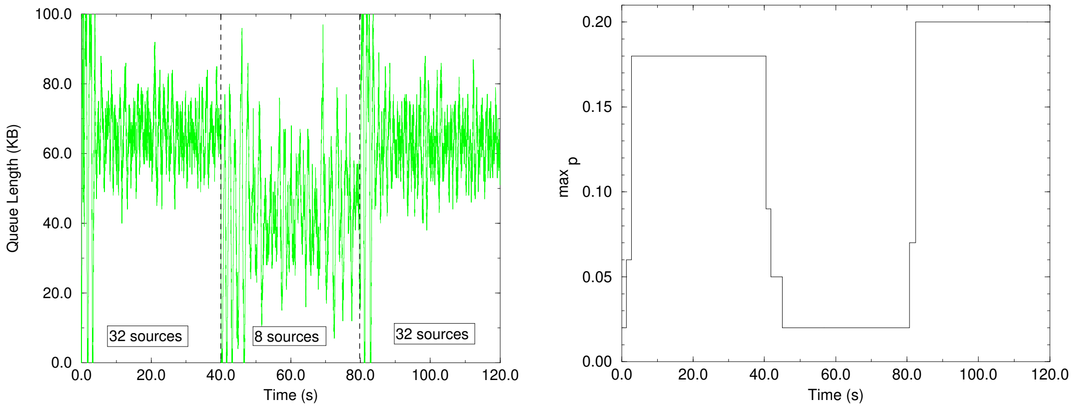
(a) Aggressive detection

(max_p = 0.250)

(b) Conservative detection

(max_p = 0.016)

Figure 3.8: Static random early detection



(a) Queue length

(b) max_p parameter

Figure 3.9: Adaptive RED

light congestion, the marking/dropping probabilities remain quite low until the average queue length reaches max_{th}. In times of heavy congestion, the marking/dropping probabilities increase quickly as the average queue length exceeds min_{th}.

To show the feasibility of Adaptive RED, another experiment using the same network shown in Figure 3.1 was run, but with RED queues of size $100KB$. In this experiment, the number of active connections is varied between 8 and 32 over 40 second intervals. Figure 3.8 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 3.8(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 where the queue is empty. When conservative early detection is used, as shown in Figure 3.8(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 3.9(a) shows the queue length plot of the same experiment using Adaptive RED with α and β set to 3 and 2, respectively. max_p is initially set to to 0.020 and then allowed to vary according to the algorithm. 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 3.9(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.

3.2.4 Round-trip time sensitivity

In the previous experiments, the round-trip time for all of the connections was kept approximately the same. While this allows for an initial understanding of the problems with RED and the effectiveness of Adaptive RED, a more realistic topology with varying round-trip times can provide slightly different results. In this section, the experimental setup is changed to reflect the presence of connections with a variety of round-trip times going across the bottleneck link. Figure 3.10 shows the network evaluated. In this network, the round trip times are varied by increasing and decreasing the transmission delay across several links. Given this heterogeneity, the round-trip delays between connections when the network is unloaded ranges from $32ms$ to $230ms$.

Using this network, the experiment in the previous section was repeated. Figure 3.11 shows the queue length traces using both aggressive and conservative early detection. The figure shows similar results as before with one small difference. When conservative early detection is used with a large number of sources, deterministic congestion notification does not always cause underutilization of the bottleneck link. One of the reasons behind this is the fact that the varying round-trip times cause sources to react to deterministic congestion notification at different times. Thus, they do not synchronously reduce their transmission rates. Another reason why underutilization is not severe is that sources with extremely small round-trip times (i.e. $32ms$) are able to ramp up their transmission rates quickly after congestion occurs. While this allows such flows to grab a disproportionate amount of the bottleneck link capacity, it also keeps the bottleneck link more fully utilized across time.

Figure 3.12 shows the performance of Adaptive RED over the same experiment. As the figure shows, the ability for Adaptive RED to hit RED's "sweet spot" is not severely affected by the difference in round-trip time. The max_p modification still allows RED to adapt effectively to changes in network load.

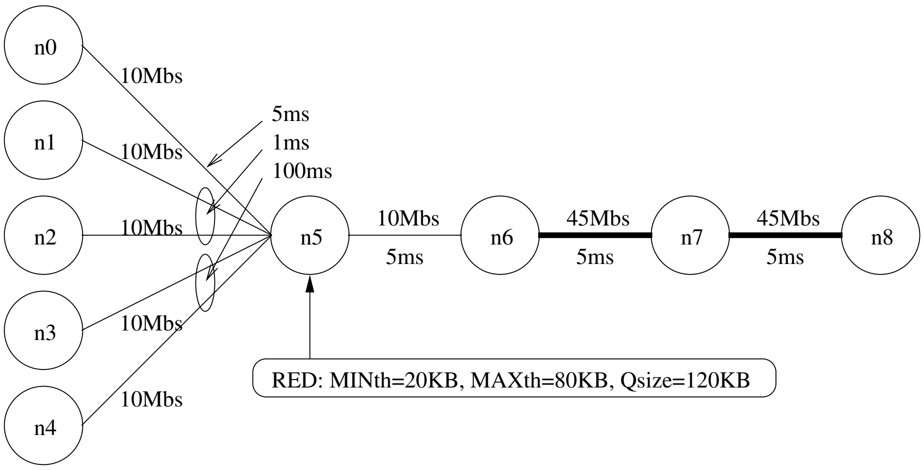
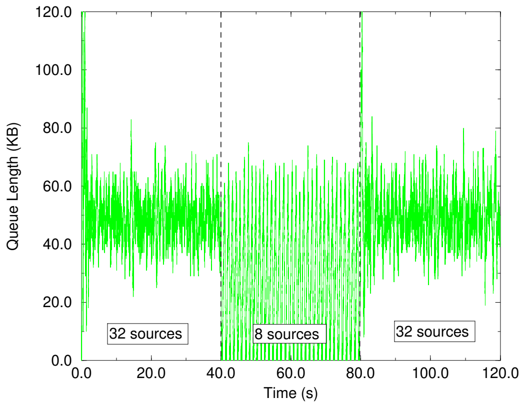
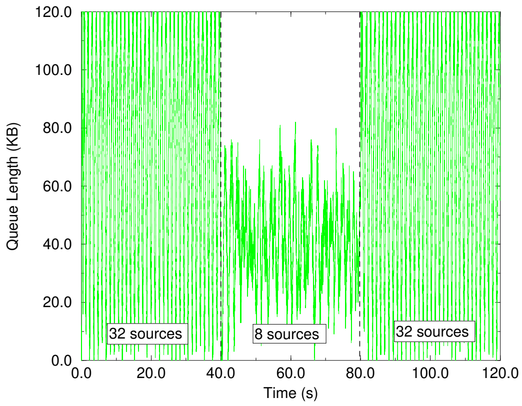


Figure 3.10: Network topology with varying round-trip times



(a) Aggressive detection

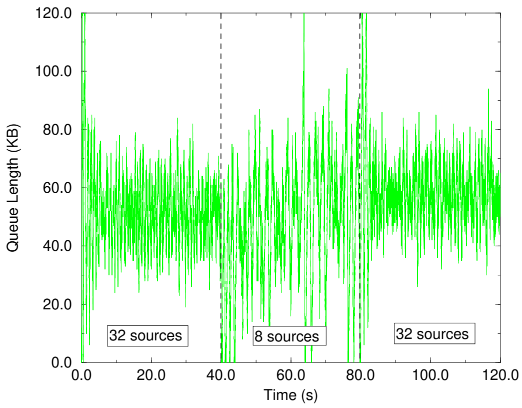
(max_p = 0.250)



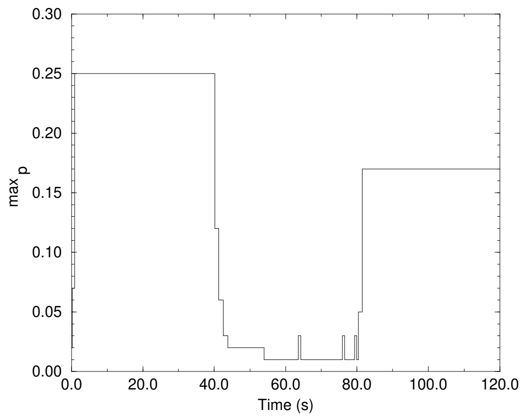
(b) Conservative detection

(max_p = 0.016)

Figure 3.11: Static random early detection over varying round-trip times



(a) Queue length



(b) max_p parameter

Figure 3.12: Adaptive RED over varying round-trip times

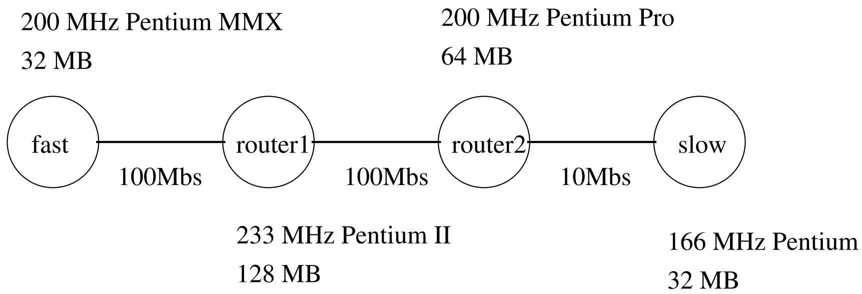
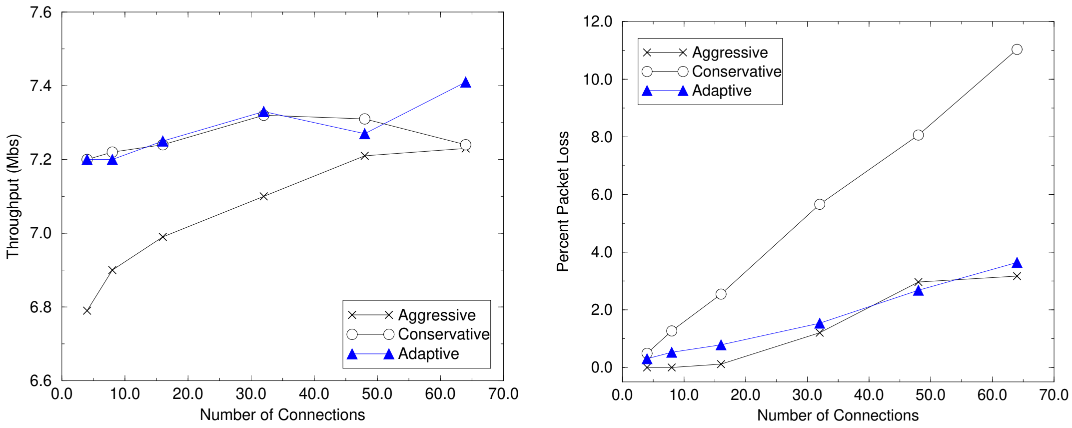


Figure 3.13: Testbed

3.2.5 Implementation

To further evaluate Adaptive RED, it has been implemented in FreeBSD 2.2.6 and ALTQ [9]. In the implementation, calls to the generic `IF_ENQUEUE` and `IF_DEQUEUE` macros from `if_output` and `if_start` are changed to replace the FIFO queueing mechanism typically used in BSD Unix with the Adaptive RED queueing discipline. Using this implementation, several experiments on a small testbed of PCs shown in Figure 3.13 were performed. In the figure, 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 on the forward path. As the figure shows, FreeBSD-based routers using Adaptive RED connect the Ethernet and Fast Ethernet segments.

To generate load on the system, `netperf` [49] is used. `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, a variable number of infinite `netperf` sessions are run from `fast` to `slow`, the endpoints of the network. The router queue on the congested interface on `router2` to the 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 Adaptive RED is initially set at 0.02 and allowed to vary according to the algorithm in Figure 3.6. In order to ensure the Adaptive RED modifications did not create bottlenecks in the routers, a number of experiments were run between `fast` and `router2` using Adaptive RED 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 Ethernet segment on `router2`.

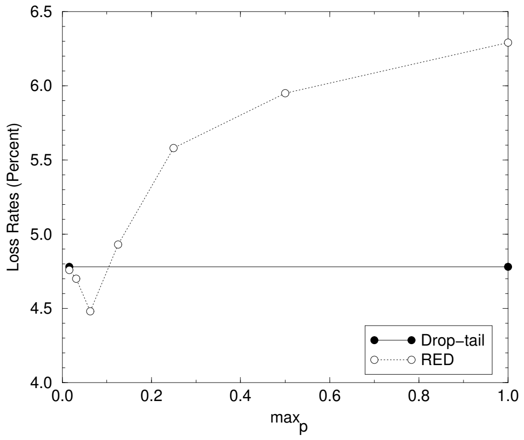


(a) Throughput

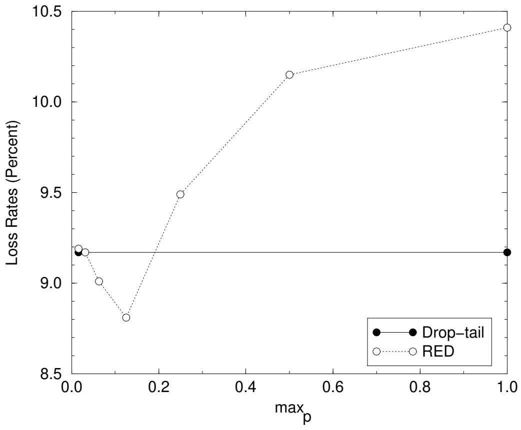
(b) Percent packet loss

Figure 3.14: Queue management performance

Figure 3.14 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 3.14(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 3.14(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 increases. When the number of connections is large, 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



(a) 32 connections



(b) 64 connections

Figure 3.15: Impact of early detection aggressiveness on RED

and link underutilization. This observation is revisited in Section 3.3 and in Chapter 4.

3.2.6 Using packet loss for congestion notification

The previous experiments examine the use of early detection in its "purest" form where congestion notification is free and causes no packet losses. Without support for ECN, however, RED must resort to dropping a packet in order 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. To illustrate this, the experiments in Section 3.2.1 were repeated using a normal RED queue. Figure 3.15 shows the loss rates of RED with 32 and 64 connections running through the bottleneck link. Again, as max_p decreases, the performance of RED approaches a drop-tail queue. However, as max_p increases, the drops contributed by the early detection algorithm begin to dominate the loss rate. Both graphs show a cusp at the point which minimizes the sum of the losses contributed by the early detection algorithm and by buffer overflow. This cusp occurs at different values of max_p depending on how many connections are present. As more connections are added, the optimal value of max_p increases.

Note that even with a RED queue parameterized to minimize loss, packet loss rates continue to increase with traffic load. Using drops as a means for congestion notification fundamentally limits the loss rate observed across the Internet. As more connections come on-line, the rate of congestion notification, and thus, the loss rates increase. Steady state analysis of the TCP congestion avoidance algorithm [22, 40, 42, 51] 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:

$$BW < \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{3.1}$$

where MSS is the segment size, RTT is the round-trip time, and C is a constant. Using this model, packet loss rates over a single bottleneck link of $L\ Mbs$ can be approximated 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 ($\frac{L}{N}$). By substituting this into the previous equation and solving for p, the following is obtained

$$p < \left(\frac{N}{L}\frac{MSS * C}{RTT}\right)^2 \tag{3.2}$$

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 phenomenon can be shown using an idealized example. Suppose two identical networks have bandwidth-delay products of $64KB$ from a given pair of end points as shown in Figure 3.16. In one network, 4 identical connections are active while in another 8 identical connections are. 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 $\frac{0.5\ packets}{RTT}$. Similarly, using the same idealized model, it can be shown that when 8 connections are present, losses occur at a rate of $\frac{2\ packets}{RTT}$, a quadratic increase.

Because the derivation of Equation 3.2 is based on idealized scenarios, the actual loss rates

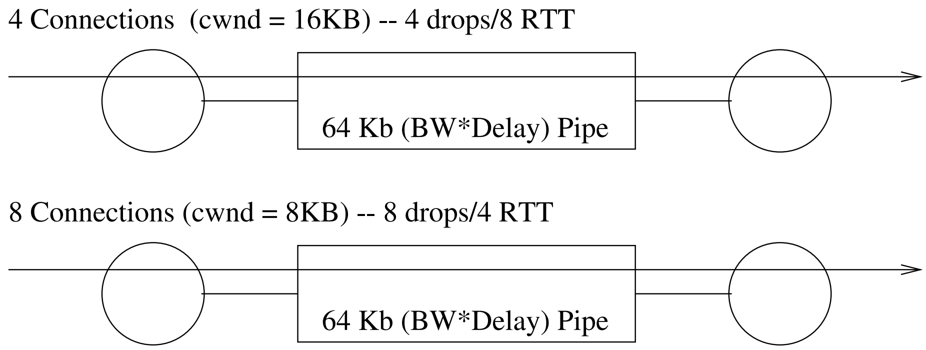


Figure 3.16: Example network

do not quite vary quadratically with the number of connections. From the drop-tail experiments in Figure 3.15, the loss rates observed show a dependence on the number of connections which is somewhere between linear and quadratic. There are several reasons why this is the case. One reason is that the derivation assumes a fair sharing of bandwidth across the bottleneck link. It has been shown that as the number of TCP connections increases, the amount of unfairness between connections increases considerably [48]. Another reason is that the equation does not model the occurrence of retransmission timeouts. By using models of TCP behavior which capture retransmission timeouts, loss rates can be predicted more accurately [52].

The results from Equation 3.2 are still significant because they show that as networks become more congested, packet loss rates increase considerably, thus making the probability of congestion collapse more likely. The equation highlights the need for decoupling packet loss from congestion notification through the use of explicit congestion notification. In lieu of explicit congestion notification, however, the equation also provides some insight on how to improve loss rates given a fixed number of connections. One way is to alter the congestion avoidance mechanism itself so that it does not continually increase its sending rate beyond the network's capacity. For example, schemes such as Tri-S [66, 67] or TCP Vegas [7] can be used to reduce the amount of packet losses observed. Another way is to increase the bottleneck link bandwidth L. By increasing the link bandwidth, the effective congestion windows of individual connections increases, thus decreasing the frequency of congestion notification in the scenario above. Reducing the segment size used in the congestion avoidance algorithm can also improve loss rates. The smaller segment size allows the end host to grow its congestion window more slowly, thus decreasing the rate at which it receives congestion notification. Finally, loss rates can be improved by increasing the round-trip time. Increasing the

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

Figure 3.17: SUBTCP algorithm

round-trip time, through the use of additional network buffers [65] increases the bandwidth delay product which slows the rate of congestion notification.

3.3 End Host Congestion Control

While properly designed active queue management mechanisms like Adaptive RED can help reduce packet losses, such techniques alone cannot guarantee low loss rates especially when traffic load fluctuates wildly. Instead, intelligent queue management must be combined with intelligent end host congestion control in order to obtain high utilization with a minimal amount of packet loss. If the offered load overwhelms the bottleneck link before congestion notification can be delivered, buffer overflow and packet loss is inevitable. This section describes several weaknesses in TCP congestion control and how they can cause high loss rates even in the presence of active queue management. Given this behavior, a number of possible modifications to TCP's windowing mechanism which alleviate this problem are proposed and evaluated.

3.3.1 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. When a large number of connections are multiplexed over a low bandwidth link, the inability of TCP to transmit at lower rates causes a significant amount of packet loss and retransmission timeouts [48]. TCP, by exponentially backing off its retransmission timeout interval, does in fact have a mechanism which allows it to transmit at a rate lower than one segment per round-trip time. However, upon a single successful transmission, TCP resets this timer and resumes transmitting at a minimum rate of one segment per round-trip time. From the standpoint of transmission rates, this in effect makes TCP congestion control an exponentially-increasing algorithm. If a large number of connections increase their transmission rates in this manner, the network sees substantial bursts which cause packet loss. One way to fix this problem is to simply 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 sending rate as soon as it has received a single acknowledgment after a retransmission timeout. Figure 3.17 shows a simple algorithm for doing so. In this case, instead of resetting TCP's back-off interval, it is halved. In the rest of the chapter, 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 [1]. Smaller segment and minimum window sizes translate into lower minimum sending rates. In addition, from Equation 3.2 in Section 3.2, using a smaller segment size makes linear increase less aggressive and thus reduces the amount of packet losses.

To understand the impact that the segment size, the minimum window size, and the SUBTCP modifications have on packet loss rates in congested networks, a number scenarios in the network shown in Figure 3.18 are examined. In this network, 100 connections are run from nodes $n0$–$n4$ through the bottleneck $1.5Mbs$ link between node A and node B to nodes $n5$–$n9$. 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}). For most of the experiments in this section, early detection is made conservative by fixing max_p to 0.016, in order to isolate the impact end host modifications have on performance. Figure 3.19(a) shows the packet loss rates observed by TCP and SUBTCP for different segment sizes and initial

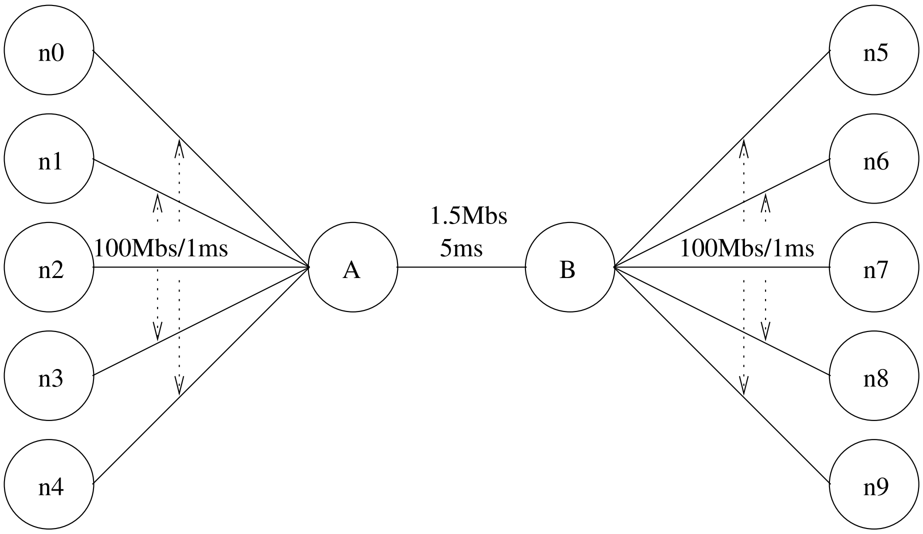
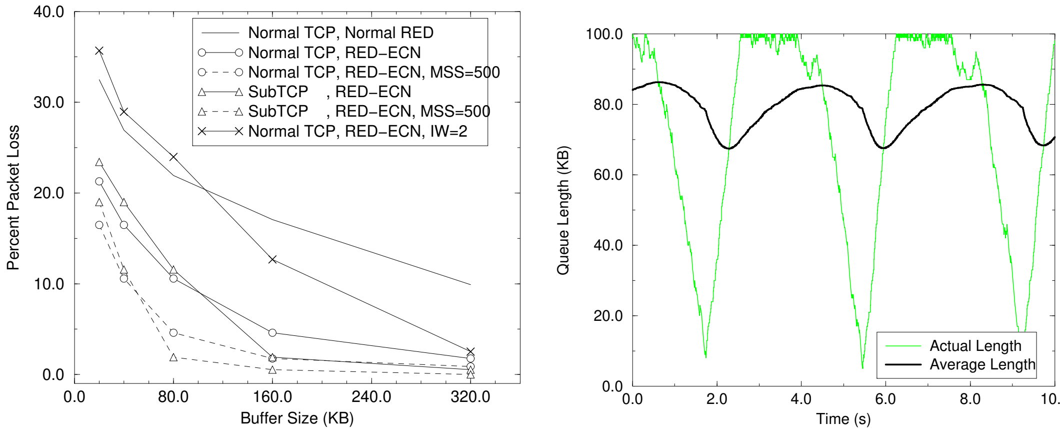


Figure 3.18: Network topology

windows. As the Figure 3.19 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, the doubling of TCP's minimum window leads to a significant increase in the amount of packet loss observed. For a better understanding of the above observations, the queue length plots of the bottleneck link for the various experiments are examined. Figure 3.19(b) shows the queue length plot of the experiment using TCP with RED-ECN queues[1]. The max_{th} used in the experiment was $80KB$. The figure shows that the queue quickly fills up soon after congestion notification stops being delivered. This is due to the fact that TCP ramps 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 3.20(a) shows the queue length plot using the smaller segment size and the SUBTCP modifications. As the figure shows, the 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. One of the reasons why is that while SUBTCP can reduce the minimum sending rates of the sources, it still allows a multiplicative increase in sending rates. Such traffic easily defeats early detection causing packet loss to occur. Additional modifications which address this are described in Section 3.3.2.

[1]The queue length plot of normal TCP over normal RED queues shows results similar to Figure 3.3(b).



(a) Packet loss rates

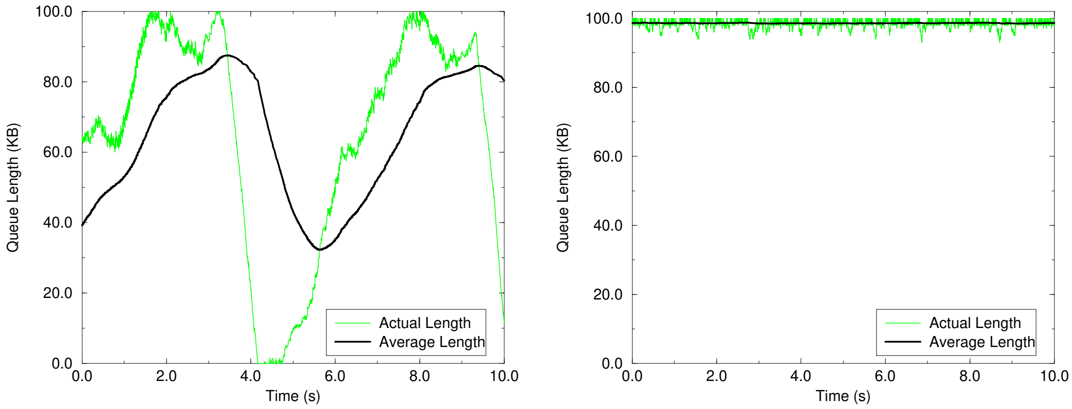
(b) Queue plot of TCP with RED-ECN

Figure 3.19: Minimum sending rates and the performance of TCP

Figure 3.20(b) shows the queue length plot of normal TCP over RED-ECN queues using a minimum window of 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. Thus, while a large minimum window may be desirable in unloaded networks where it can reduce transfer latencies considerably, it must be used carefully or not at all when the network is congested.

3.3.2 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 is multiplicatively increasing and decreasing. This causes rapid fluctuations in queue lengths, periods 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.



(a) SUBTCP with MSS=500 (b) TCP with $cwnd_{min}$=2

Figure 3.20: Queue plots for decreased segment size and increased window size

In this section, modifications to the linear increase algorithm are considered in order to enable TCP to work better under heavy congestion. Two techniques, in particular, are considered: (1) a scaled linear increase and (2) a 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 not being aggressive enough when the network is uncongested or being too aggressive when the network is congested. Figure 3.21 shows the scaled linear increase algorithm used. In the experiments, scaled increases are used 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, an experimental, bandwidth-based linear increase algorithm was also examined. The motivation of the bandwidth-based algorithm is rather intuitive. Assume that the bottleneck router has enough buffers to absorb $X\%$ higher than its bottleneck 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 *at most* $X\%$ 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 inherently de-

```
opencwnd()
        if (CWND == 1 && scale > 1)
                scale = scale - scale_factor;
                if (scale < 1) scale = 1;
        else normal_tcp_opencwnd();
```

Figure 3.21: Scaled sub-linear SUBTCP algorithm

pend on the amount of buffering at the bottleneck link. If the buffers are increased at the bottleneck link, the end sources can be more aggressive in ramping up their transmission rates. On the other hand, increased buffering can also prevent sources from ramping up their transmission rates by increasing the round-trip time and thus the latency in delivering congestion notification. 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 average queue length 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 is that TCP traffic exhibits short-term burstiness which makes its offered load appear to be much larger than it actually is [70]. 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 3.22 shows the bandwidth-based linear increase algorithm used in the 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. Note that one of the disadvantages of the bandwidth-based increase algorithm is that it falls in a class of algorithms which have been theoretically shown to be unable to provide max-min fair sharing in a responsive manner [8, 38]. 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 [25, 48]. In addition, the idealized model used in [8] 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 [26, 41] that deliver

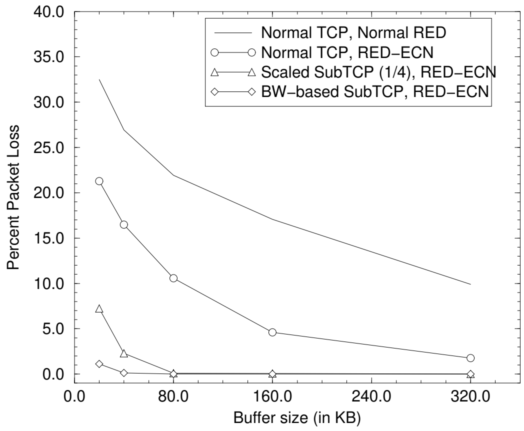
opencwnd()
  e_cwnd = $\frac{\text{CWND}}{scale}$;
  wnd = min(e_cwnd $\times$ $\frac{(1+X)}{\text{CWND}}$, e_cwnd + $\frac{1}{\text{CWND}}$);
  if (wnd $<$ 1)
    CWND = 1;
    $scale$ = $\frac{1}{wnd}$;
  else
    CWND = wnd;
    $scale$ = 1;

Figure 3.22: Bandwidth-based SUBTCP algorithm

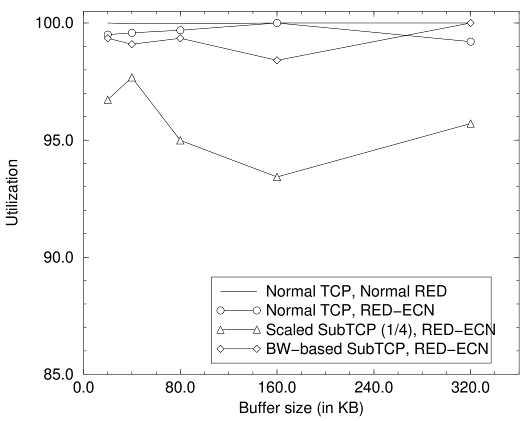
congestion notification preferentially to higher bandwidth flows, it may be possible for a larger class of increase/decrease algorithms to quickly converge to max-min fair sharing.

Given these modified increase algorithms, the previous experiments were repeated in order to evaluate their performance. Figure 3.23 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, the percent bandwidth increase was set to 1% in order to prevent packet loss in the network shown in Figure 3.18 when using a $25KB$ RED queue with a min_{th} of $5KB$ and a max_{th} of $20KB$. Figure 3.23(a) shows the loss rates observed using the various TCP schemes. The graph shows that using the sub-linear increases greatly reduces the amount of packet loss. In fact, for smaller buffer sizes, the difference 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 3.23(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 utilization across all buffer sizes.

In order to understand why the scaled linear increase algorithm sees lower link utilization and sometimes higher packet loss than the bandwidth-based algorithm, the queue length plots of the bottleneck queue were captured. Figure 3.24(a) plots the queue lengths from the experiment using a max_{th} of $80KB$. The traces show that the use of sub-linear increases makes aggregate TCP less aggressive, thus preventing sustained periods of packet loss observed in the previous experiments shown in Figure 3.20. Figure 3.24(a) also shows, however, that the sources are still aggressive enough to defeat the early detection mechanism. It takes a sustained period of congestion notifica-



(a) Packet loss rates



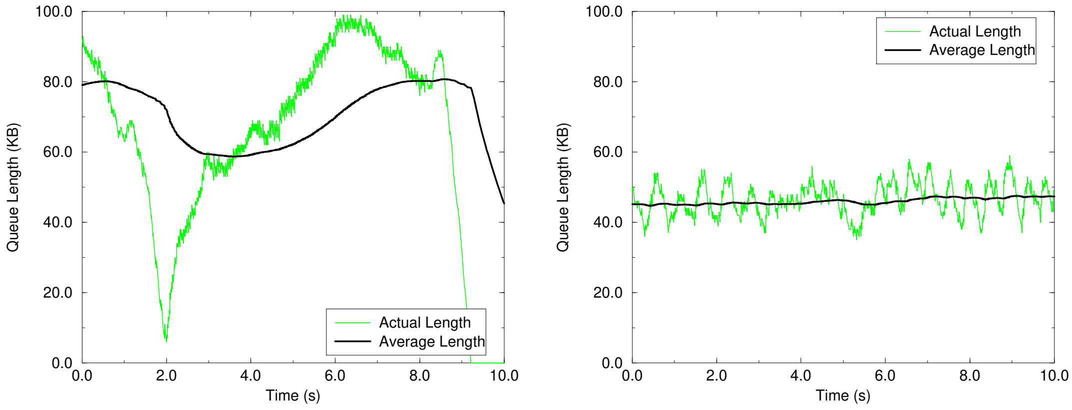
(b) Bottleneck link utilization

Figure 3.23: Performance of modified linear-increase algorithms

tion when the average queue length exceeds max_{th} for the offered load to be reduced sufficiently below link capacity. As shown in Section 3.2.2, sustained congestion notification can be detrimental to link utilization since it inevitably causes too many sources to back off their transmission rates. As the queue plot shows, when max_{th} is triggered, the bottleneck link can subsequently become underutilized. There are a couple of ways to fix 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 3.2. Additional experiments have shown both ways can be effective in improving the performance of scaled increases. In contrast to the scaled linear increase, Figure 3.24(b) shows the queue length plot when each source is using bandwidth-based increases. As the figure shows, the aggregate traffic fluctuates much less, allowing early detection to perform as intended.

3.4 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 3.25 shows the loss rates and bottleneck link utilizations over a range of workloads using the bandwidth-based SUBTCP algorithm with Adaptive RED queues in



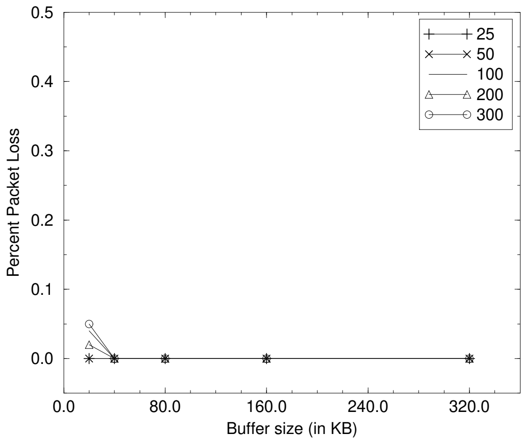
(a) Scaled increases (factor = $\frac{1}{4}$) (b) Bandwidth-based increases

Figure 3.24: Sub-linear SUBTCP performance

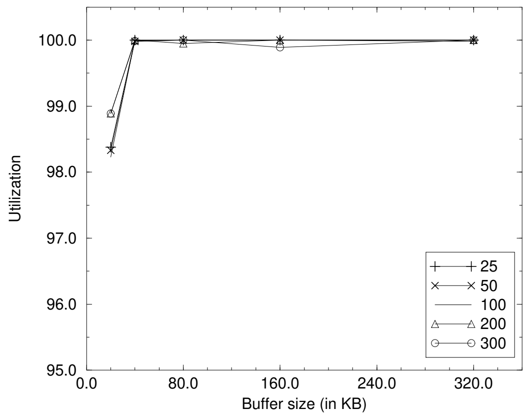
the network. In these experiments, the topology in Figure 3.18 is used and the number of connections going across the $1.5Mbs$ link is varied from 25 to 300. As the figure shows, the packet loss rates remain extremely low while the bottleneck link utilizations remain remarkably high across all experiments.

In order to compare the improvements against other schemes, the packet loss rates using a traffic load of 100 connections is used. Figure 3.26(a) plots the loss rates for a range of end host and queue management schemes. The figure shows the performance of normal TCP using both drop-tail and RED queues as well as the performance of normal TCP using ECN using 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. In this figure, the graph shows that decoupling congestion notification from packet loss through the use of ECN improves loss rates significantly. The graph also shows that both the Adaptive RED and SUBTCP modifications provide substantial performance improvement and that, when used together, they allow the network to achieve optimal performance.

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, additional experiments which varied them were performed. Figure 3.26(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 $5ms$ to $10ms$ and $50ms$, thus considerably increasing the round-trip time. As the figure shows, as each



(a) Packet loss rates



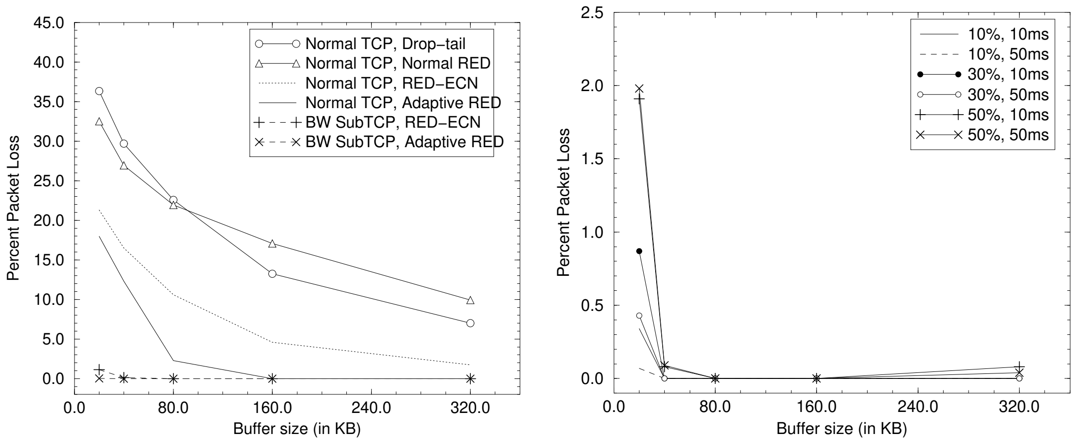
(b) Bottleneck link utilization

Figure 3.25: Performance across traffic load

source is allowed to increase its sending rate more quickly, loss rates slowly rise as the burstiness seen at the router increases. However, even with larger percent increases, 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, with loss rates remaining low as well.

3.5 Conclusion and Future Work

This chapter 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 both provide significant benefits in terms of decreasing packet loss and increasing network utilization. There are several ways in which these mechanisms can be extended. In particular, several ways for methodically improving the adaptiveness of the RED algorithm are being examined. This chapter 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 could actually keep track of the number of active connections and change its aggressiveness accordingly. Another mechanism would be to have the RED queue infer the number of connections present by the rate at which the average queue length changes and have it then adapt its parameters accordingly. It may also be possible to adapt other RED parameters instead of max_p



(a) Comparison between schemes

(b) Parameter and RTT sensitivity

Figure 3.26: Performance comparisons

to optimize performance. For example, one could adaptively change inter-packet drop probabilities or the RED threshold values depending on the input traffic. Finally, in extremely congested networks for which setting max_p to 1 is not sufficient, it may be possible to make the marking/dropping even more aggressive by having the marking probability change as a non-linear function of the average queue length. Figure 3.27 shows an example of how the marking/dropping function can be modified to further improve the performance of RED. As the figure shows, when a max_p setting of 1 is insufficient, the marking function can assume a non-linear shape allowing the marking to become even more aggressive. Changing the marking function in this manner can allow RED to control congestion even under extremely heavy congestion.

Additional ways for improving end host congestion control algorithms are also being examined. While bandwidth-based increases provide end hosts with an upper bound on how aggressively they can ramp up their sending rates, it is often desirable for a source to change its sending rate more slowly or not at all when nearing the congestion point in order to avoid oscillations inherent in TCP's windowing algorithm [7, 66, 67]. Incorporating such techniques into the bandwidth-based increase algorithm are being explored. Additional mechanisms for improving fairness over short time scales are being considered. One of the advantages of TCP congestion avoidance is that it inherently gives an advantage to low-bandwidth flows. For flows with equivalent round-trip times, low-bandwidth flows with small congestion windows increase their effective sending rates more quickly than higher-bandwidth flows with larger windows when using TCP's congestion-avoidance

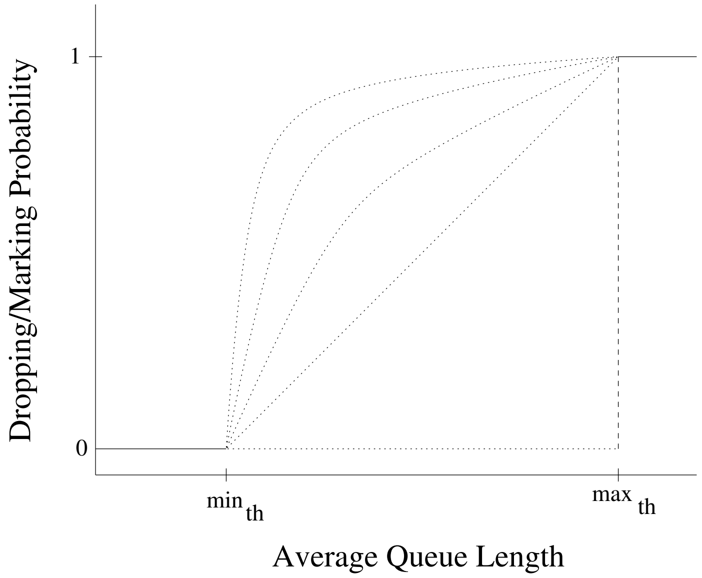


Figure 3.27: Extension to Adaptive RED

algorithm. Extensions which address this problem in the context of bandwidth-based increases are also being considered.

CHAPTER 4

BLUE: A NEW CLASS OF ACTIVE QUEUE MANAGEMENT ALGORITHMS

4.1 Introduction

One of the striking results in Chapter 3 is that even with ECN, RED queue management is still unable to eliminate packet loss over a large range of workloads. Packet loss can only be eliminated when modifications are made to TCP's congestion control algorithms. This chapter demonstrates a fundamental weakness with RED and all other known active queue management techniques. This weakness severely impacts their ability to minimize packet loss. The problem lies in the fact that all of the algorithms rely on some form of the queue length in order to estimate congestion. While the presence of a persistent queue indicates congestion, its length gives very little information as to the severity of congestion, that is, the number of competing connections sharing the link. In a busy period, a single source transmitting at a rate greater than the bottleneck link capacity can cause a queue to build up just as easily as a large number of sources can. Since the RED algorithm relies on queue lengths, it has an inherent problem in determining the severity of congestion. As a result, RED requires a wide range of parameters to operate correctly under different congestion scenarios. While RED can achieve an ideal operating point, it can only do so when it has a sufficient amount of buffer space and is correctly parameterized [15, 65].

In light of the above observation, this chapter proposes a fundamentally different active queue management algorithm, called BLUE, which uses packet loss and link utilization history to manage congestion. BLUE maintains a single probability, which it uses to mark (or drop) packets when they are queued. If the queue is continually dropping packets due to buffer overflow, BLUE increments the marking probability, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability. Using simulation and experimentation, this chapter demonstrates the superiority of BLUE to RED in reducing packet losses even when operating with a smaller buffer. BLUE maintains low packet loss rates over a wide-range of workloads *without* requiring modifications to TCP's congestion control algorithm. Finally, using mechanisms based on BLUE, this chapter proposes and evaluates Stochastic Fair BLUE (SFB), a novel mechanism for effectively and scalably enforcing fairness amongst a large number of flows.

4.2 The Inadequacy of RED

As described in Chapter 2, one of the biggest problems with TCP's congestion control algorithm over drop-tail queues is that the sources reduce their transmission rates only after detecting packet loss due to queue overflow. Since considerable amount of time may elapse between the packet drop at the router and its detection at the source, a large number of packets may be dropped as the senders continue transmission at a rate that the network cannot support. RED alleviates this problem by detecting incipient congestion *early* and delivering congestion notification to the end hosts, allowing them to reduce their transmission rates before queue overflow occurs. In order to be effective, a RED queue must be configured with a sufficient amount of buffer space to accommodate an applied load greater than the link capacity from the instant in time that congestion is detected using the queue length trigger to the instant in time that the applied load decreases at the bottleneck link in response to congestion notification. RED must also ensure that congestion notification is given at a rate which sufficiently suppresses the transmitting sources without underutilizing the link. Unfortunately, when a large number of TCP sources are active, the aggregate traffic generated is extremely bursty [18, 21]. Bursty traffic often defeats the active queue management techniques used by RED since queue lengths grow and shrink rapidly, well before RED can react. Figure 4.1 shows a simplified pictorial example of how RED functions under this congestion scenario.

The congestion scenario presented in Figure 4.1 occurs when a large number of TCP sources are active and when a small amount of buffer space is used at the bottleneck link. As the figure

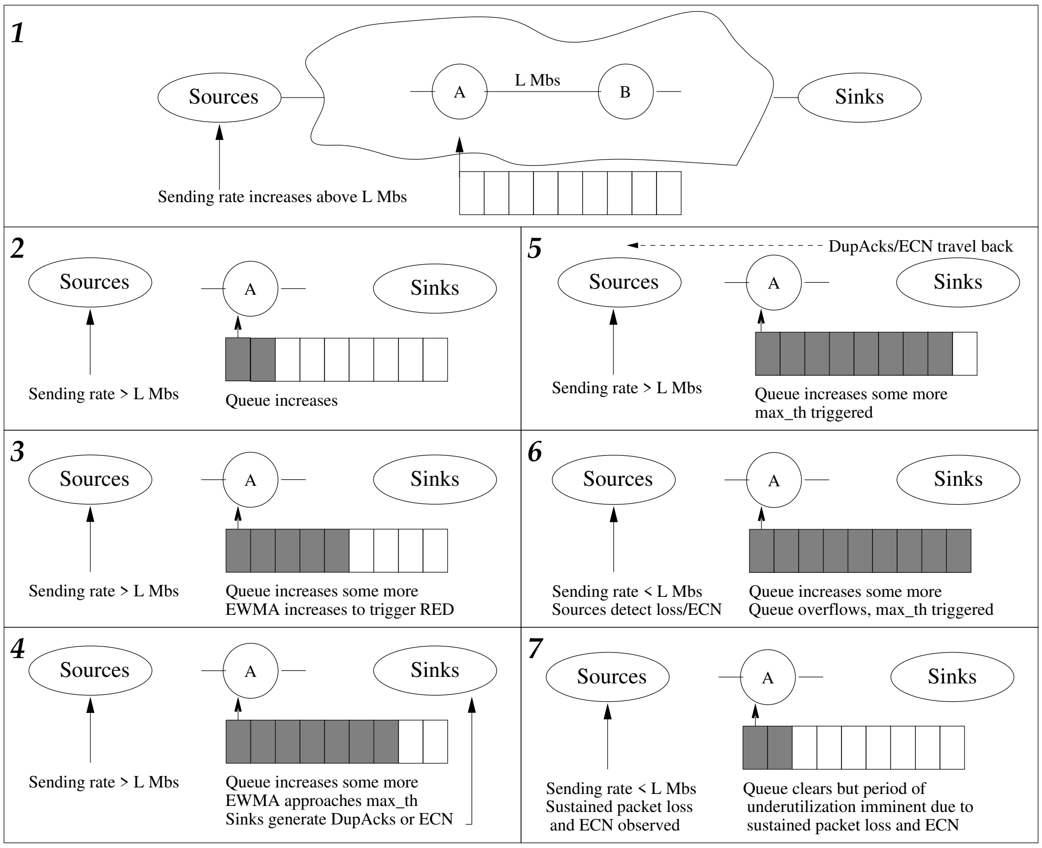


Figure 4.1: RED example

shows, at $t = 1$, a sufficient change in aggregate TCP load (due to TCP opening its congestion window) causes the transmission rates of the TCP sources to exceed the capacity of the bottleneck link. At $t = 2$, the mismatch between load and capacity causes a queue to build up at the bottleneck. At $t = 3$, the average queue length exceeds min_{th} and the congestion-control mechanisms are triggered. At this point, congestion notification is sent back to the end hosts at a rate dependent on the queue length and marking probability max_p. At $t = 4$, the TCP receivers either detect packet loss or observe packets with their ECN bits set. In response, duplicate acknowledgments and/or TCP-based ECN signals are sent back to the sources. At $t = 5$, the duplicate acknowledgments and/or ECN signals make their way back to the sources to signal congestion. At $t = 6$, the sources finally detect congestion and adjust their transmission rates. Finally, at $t = 7$, a decrease in offered load at the bottleneck link is observed. Note that it has taken from $t = 1$ until $t = 7$ before the offered load becomes less than the link's capacity. Depending upon the aggressiveness of the aggregate TCP sources [18, 21] and the amount of buffer space available in the bottleneck link, a

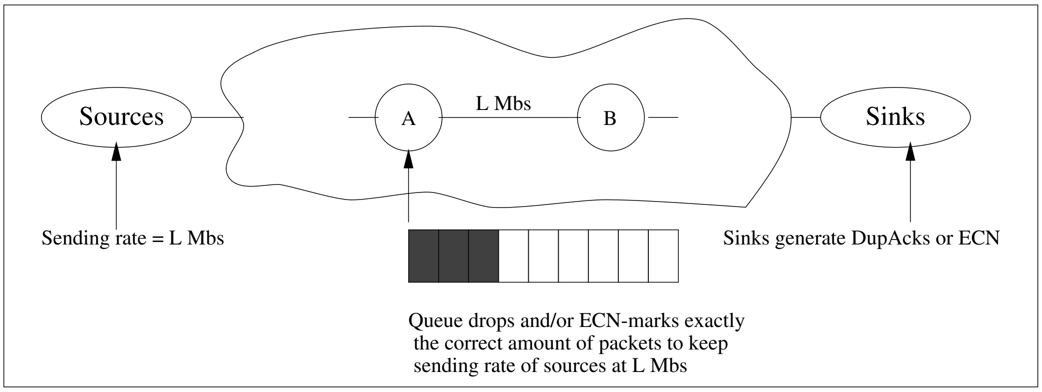


Figure 4.2: Ideal scenario

large amount of packet loss and/or deterministic ECN marking may occur. Such behavior leads to eventual underutilization of the bottleneck link.

One way to solve this problem is to use a large amount of buffer space for the RED queue. For example, it has been suggested that in order for RED to work well, an intermediate router requires buffer space that amounts to twice the bandwidth-delay product [65]. This approach, in fact, has been taken by an increasingly large number of router vendors. Unfortunately, in networks with large bandwidth-delay products, the use of a large amount of buffer adds considerable end-to-end delay and delay jitter. This severely impacts the ability to run interactive applications. In addition, the abundance of deployed routers which have limited memory resources makes this solution undesirable.

Figure 4.2 shows how an ideal queue management algorithm works. In this figure, the congested gateway delivers congestion notification at a rate which keeps the aggregate transmission rates of the TCP sources at or just below the clearing rate. While RED can achieve this ideal operating point, it can do so only when it has a sufficiently large amount of buffer space and is correctly parameterized.

4.3 BLUE

In order to remedy the shortcomings of RED, this section proposes and evaluates a fundamentally different queue management algorithm called BLUE. Using both simulation and experimentation, BLUE is shown to overcome many of RED's shortcomings. RED has been designed with the objective to (1) minimize packet loss and queueing delay, (2) avoid global synchronization of sources, (3) maintain high link utilization, and (4) remove biases against bursty sources. This sec-

```
Upon packet loss event:
    if ( (now - last_update) > freeze_time ) then
        p_m = p_m + delta
        last_update = now
Upon link idle event:
    if ( (now - last_update) > freeze_time ) then
        p_m = p_m - delta
        last_update = now
```

Figure 4.3: The BLUE algorithm

tion shows how BLUE either improves or matches RED's performance in all of these aspects. The results also show that BLUE converges to the ideal operating point shown in Figure 4.2 in almost all scenarios, even when used with very small buffers.

4.3.1 The algorithm

The key idea behind BLUE is to perform queue management based directly on packet loss and link utilization rather than on the instantaneous or average queue lengths. This is in contrast to all known active queue management schemes which use some form of queue occupancy in their congestion management. BLUE maintains a single probability, p_m, which it uses to mark (or drop) packets when they are enqueued. If the queue is continually dropping packets due to buffer overflow, BLUE increments p_m, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability. Figure 4.3 shows the BLUE algorithm. Note that besides the marking probability, BLUE uses two other parameters which control how quickly the marking probability changes over time. The first is *freeze_time*. This parameter determines the minimum time interval between two successive updates of p_m. This allows the changes in the marking probability to take effect before the value is updated again. While the experiments in this chapter fix *freeze_time* as a constant, this value should be randomized in order to avoid global synchronization [25]. The other parameter used, *delta*, determines the amount by which p_m is incremented when the queue overflows, or decremented when

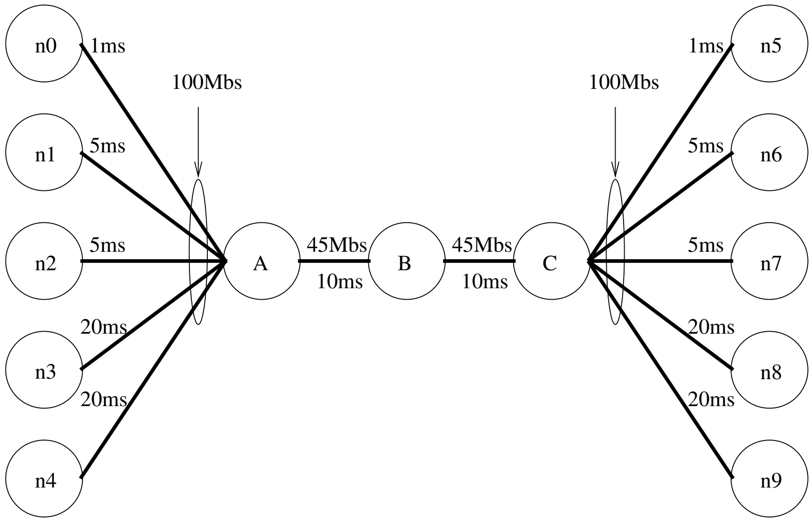
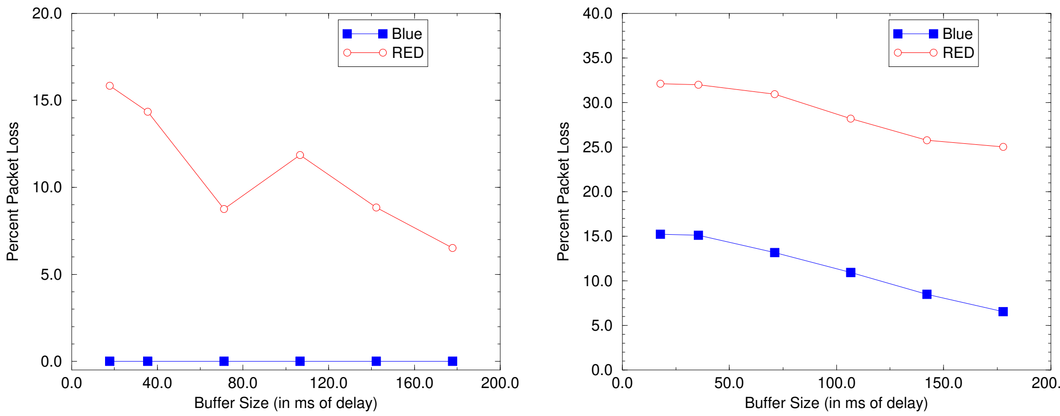


Figure 4.4: Network topology

the link is idle. Note that there are a myriad of ways in which p_m can be managed. Experiments using a wide range of parameter settings and algorithm variations have also been performed with the only difference being how quickly the queue management algorithm adapts to the offered load. While BLUE seems extremely simple, it provides a significant performance improvement even when compared to a RED queue which has been optimally parameterized.

4.3.2 Packet loss rates using RED and BLUE

In order to evaluate the performance of BLUE, a number of experiments were run using `ns` [46] over a small network shown in Figure 4.4. Using this network, Pareto on/off sources with mean on-times of 2 seconds and mean off-times of 3 seconds were run from one of the leftmost nodes (n_0, n_1, n_2, n_3, n_4) to one of the rightmost nodes (n_5, n_6, n_7, n_8, n_9). In addition, all sources were enabled with ECN support and were randomly started within the first second of simulation. Packet loss statistics were then measured after 10 seconds of simulation for 100 seconds. Loss statistics were also measured for RED using the same network and under identical conditions. For the RED queue, min_{th} and max_{th} were set to 30% and 90% of the queue size, respectively. RED's congestion notification mechanism was made as aggressive as possible by setting max_p to 1. For these experiments, this is the ideal setting of max_p since it minimizes both the queueing delay and packet loss rates for RED [21]. For the BLUE experiments, *delta* was set to 0.01 and *freeze_time* was set to $10ms$. Again, simulations using a range of *delta* and *freeze_time* values were also performed



(a) 1000 sources (b) 4000 sources

Figure 4.5: Packet loss rates of RED and BLUE

and showed similar results with the only difference being how quickly p_m converges to the correct value.

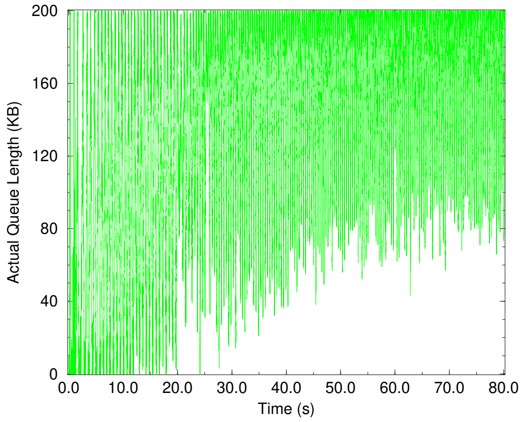
Figure 4.5 shows the loss rates observed over different queue sizes using both BLUE and RED with 1000 and 4000 connections present. In these experiments, the queue at the bottleneck link between A and B is sized from $100KB$ to $1000KB$. This corresponds to queueing delays which range from $17.8ms$ and $178ms$ as shown in the figure. In all experiments, the link remains over 99.9% utilized. As Figure 4.5(a) shows, with 1000 connections, BLUE maintains zero loss rates over all queue sizes even those which are below the bandwidth-delay product of the network [65]. This is in contrast to RED which suffers double-digit loss rates as the amount of buffer space decreases. An interesting point in the RED loss graph shown in Figure 4.5(a) is that it shows a significant dip in loss rates at a buffering delay of around $80ms$. This occurs because of a special operating point of RED when the average queue length stays above max_{th} all the time. At several points during this particular experiment, the buffering delay and offered load match up perfectly to cause the average queue length to stay at or above max_{th}. In this operating region, the RED queue marks every packet, but the offered load is aggressive enough to keep the queue full. This essentially allows RED to behave at times like BLUE with a marking probability of 1 and a queueing delay equivalent to max_{th}. This unique state of operation is immediately disrupted by any changes in the load or round-trip times, however. When the buffering delay is increased, the corresponding round-trip times increase and cause the aggregate TCP behavior to be less aggressive. Deterministic marking on this less aggressive load causes fluctuations in queue length which can increase packet loss rates since RED undermarks packets at times. When the buffering delay is decreased, the corresponding round-trip times decrease and cause the aggregate TCP behavior to be more aggressive. As a result, packet loss is often accompanied with deterministic marking. When combined, this leads again to fluctuations in queue length. At a load which is perfectly selected, the average queue length of RED can remain at max_{th} and the queue can avoid packet loss and prevent queue fluctuations by marking every packet.

As Figure 4.5(b) shows, when the number of connections is increased to 4000, BLUE still significantly outperforms RED. Even with an order of magnitude more buffer space, RED still cannot match BLUE's loss rates using $17.8ms$ of buffering at the bottleneck link. It is interesting to note that BLUE's marking probability remains at 1 throughout the duration of all of these experiments. Thus, even though every packet is being marked, the offered load can still cause a significant amount of packet loss. The reason why this is the case is that the TCP sources being used do not invoke a retransmission timeout upon receiving an ECN signal with a congestion window of 1. Section 4.3.4 shows how this can significantly influence the performance of both RED and BLUE.

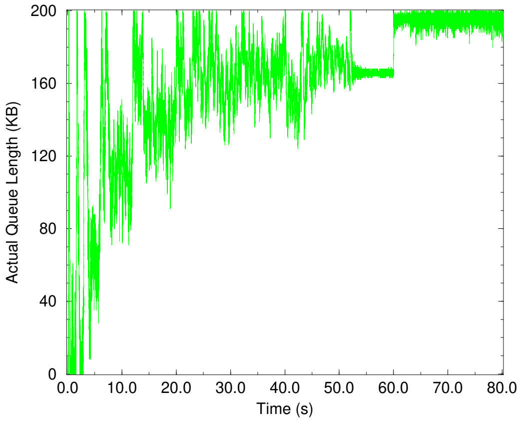
The most important consequence of using BLUE is that congestion control can be performed with a minimal amount of buffer space. This reduces the end-to-end delay over the network, which in turn, improves the effectiveness of the congestion control algorithm. In addition, smaller buffering requirements allow more memory to be allocated to high priority packets [10, 30], and frees up memory for other router functions such as storing large routing tables. Finally, BLUE allows legacy routers to perform well even with limited memory resources.

4.3.3 Understanding BLUE

To fully understand the difference between the RED and BLUE algorithms, Figure 4.6 compares their queue length plots in an additional experiment. In this experiment, a workload of infinite sources is changed by increasing the number of connections by 200 every 20 seconds. As Figure 4.6(a) shows, RED sustains continual packet loss throughout the experiment. In addition, at lower loads, periods of packet loss are often followed by periods of underutilization as deterministic packet marking and dropping eventually causes too many sources to reduce their transmission rates. In contrast, as Figure 4.6(b) shows, since BLUE manages its marking rate more intelligently, the queue length plot is more stable. Congestion notification is given at a rate which neither causes



(a) RED

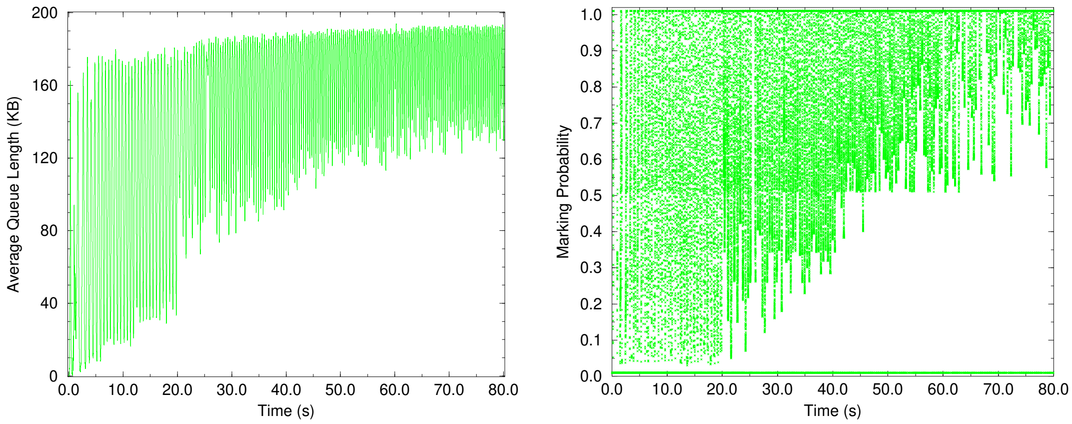


(b) BLUE

Figure 4.6: Queue length plots of RED and BLUE

periods of sustained packet loss nor periods of continual underutilization. Only when the offered load rises to 800 connections, does BLUE sustain a significant amount of packet loss.

Figure 4.7 plots the average queue length (Q_{ave}) and the marking probability ($\frac{p_b}{1-count \times p_b}$) of RED throughout the experiment. The average queue length of RED contributes directly to its marking probability since p_b is a linear function of Q_{ave} ($p_b = max_p \times \frac{Q_{ave}-min_{th}}{max_{th}-min_{th}}$). As shown in Figure 4.7(a), the average queue length of RED fluctuates considerably as it follows the fluctuations of the instantaneous queue length. Because of this, the marking probability of RED, as shown in Figure 4.7(b), fluctuates considerably as well. In contrast, Figure 4.8 shows the marking probability of BLUE. As the figure shows, the marking probability converges to a value that results in a rate of congestion notification which prevents packet loss and keeps link utilization high throughout the experiment. In fact, the only situation where BLUE cannot prevent sustained packet loss is when every packet is being marked, but the offered load still overwhelms the bottleneck link. As described earlier, this occurs at $t = 60s$ when the number of sources is increased to 800. The reason why packet loss still occurs when every packet is ECN-marked is that for these sets of experiments, the TCP implementation used does not invoke an RTO when an ECN signal is received with a congestion window of 1. This adversely affects the performance of both RED and BLUE in this experiment. Note that the comparison of marking probabilities between RED and BLUE gives some insight as to how to make RED perform better. By placing a low pass filter on the calculated marking probability of RED, it may be possible for RED's marking mechanism to behave in a manner similar to BLUE's.



(a) Q_{ave} (b) $\frac{p_b}{1-count \times p_b}$

Figure 4.7: Marking behavior of RED

While low packet loss rates, low queueing delays, and high link utilization are extremely important, the queue length and marking probability plots allow us to explore the effectiveness of RED and BLUE in preventing global synchronization and in removing biases against bursty sources. RED attempts to avoid global synchronization by randomizing its marking decision and by spacing out its marking. Unfortunately, when aggregate TCP load changes dramatically as it does when a large amount of connections are present, it becomes impossible for RED to achieve this goal. As Figure 4.7(b) shows, the marking probability of RED changes considerably over very short periods of time. Thus, RED fails to mark packets evenly over time and hence cannot remove synchronization among sources. As Figure 4.8 shows, the marking probability of BLUE remains steady. As a result, BLUE marks packets randomly and evenly over time. Consequently, it does a better job in avoiding global synchronization.

Another goal of RED is to eliminate biases against bursty sources in the network. This is done by limiting the queue occupancy so that there is always room left in the queue to buffer transient bursts. In addition, the marking function of RED takes into account the last packet marking time in its calculations in order to reduce the probability that consecutive packets belonging to the same burst are marked. Using a single marking probability, BLUE achieves the same goal equally well. As the queue length plot of BLUE shows (Figure 4.6), the queue occupancy remains below the actual capacity, thus allowing room for a burst of packets. In addition, since the marking probability remains smooth over large time scales, the probability that two consecutive packets from a smoothly

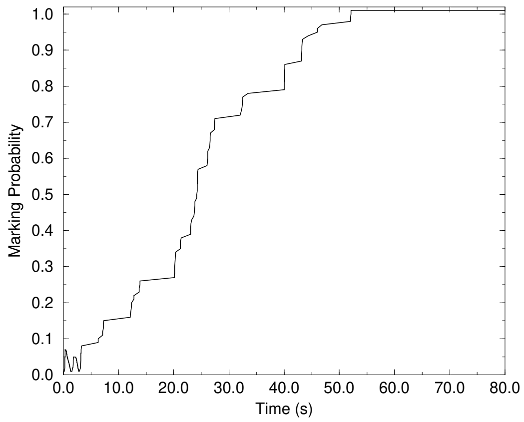


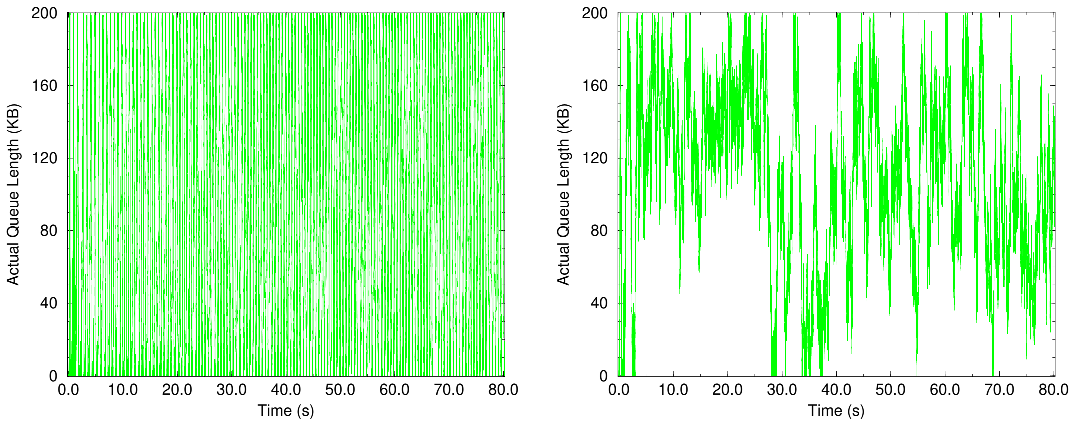
Figure 4.8: Marking behavior of BLUE (p_m)

transmitting source are marked is the same as with two consecutive packets from a bursty source.

4.3.4 The effect of ECN timeouts

All of the previous experiments use TCP sources which support ECN, but do not perform a retransmission timeout upon receipt of an ECN signal with a congestion window of 1. This has a significant, negative impact on the packet loss rates observed for both RED and BLUE especially at high loads. Figure 4.9 shows the queue length plot of RED and BLUE using the same experiment as above with TCP sources enabled with ECN timeouts. Figure 4.9(a) shows that by deterministically marking packets at max_{th}, RED oscillates between periods of packet loss and periods of underutilization as described in Section 4.2. Note that this is in contrast to Figure 4.6(a) where without ECN timeouts, TCP is aggressive enough to keep the queue occupied at all times. An interesting point to make is that RED can effectively prevent packet loss by setting its max_{th} value sufficiently far below the size of the queue. In this experiment, a small amount of loss occurs since deterministic ECN marking does not happen in time to prevent packet loss. While the use of ECN timeouts allows RED to avoid packet loss, the deterministic marking eventually causes underutilization at the bottleneck link. Figure 4.9(b) shows the queue length plot of BLUE over the same experiment. In contrast to RED, BLUE avoids deterministic marking and maintains a marking probability that allows it to achieve high link utilization while avoiding sustained packet loss over all workloads.

Figure 4.10 shows the corresponding marking behavior of both RED and BLUE in the experiment. As the figure shows, BLUE maintains a steady marking rate which changes as the workload is



(a) RED (b) BLUE

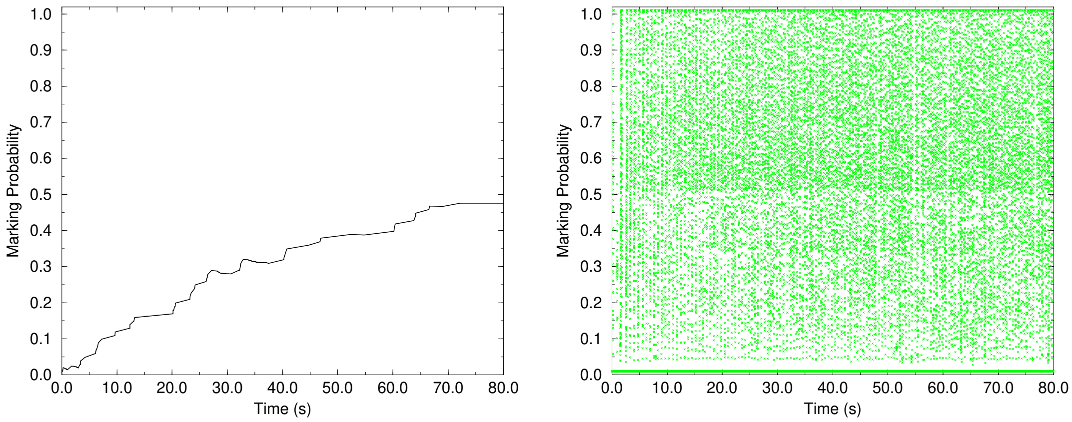
Figure 4.9: Queue length plots of RED and BLUE with ECN timeouts

changed. On the other hand, RED's calculated marking probability fluctuates from 0 to 1 throughout the experiment. When the queue is fully occupied, RED overmarks and drops packets causing a subsequent period of underutilization as described in Section 4.2. Conversely, when the queue is empty, RED undermarks packets causing a subsequent period of high packet loss as the offered load increases well beyond the link's capacity.

Figure 4.11 shows how ECN timeouts impact the performance of RED and BLUE. The figure shows the loss rates and link utilization using the 4000 connection experiments in Section 4.3.2. As the figure shows, the use of ECN timeouts allows RED to effectively prevent packet loss. However, because it often deterministically marks packets, it suffers from poor utilization. BLUE, on the other hand, maintains low packet loss rates and high link utilization across all experiments.

4.3.5 Implementation

In order to evaluate BLUE in a more realistic setting, it has been implemented in FreeBSD 2.2.7 and ALTQ [9]. The implementation was done in a manner similar to that of Adaptive RED, as described in Section 3.2.5. Using this implementation, several experiments were run on the testbed shown in Figure 4.12. Each network node and link is labeled with the 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 on the forward path. As the figure shows, FreeBSD-based routers using the BLUE queue management algorithm connect the Ethernet and Fast



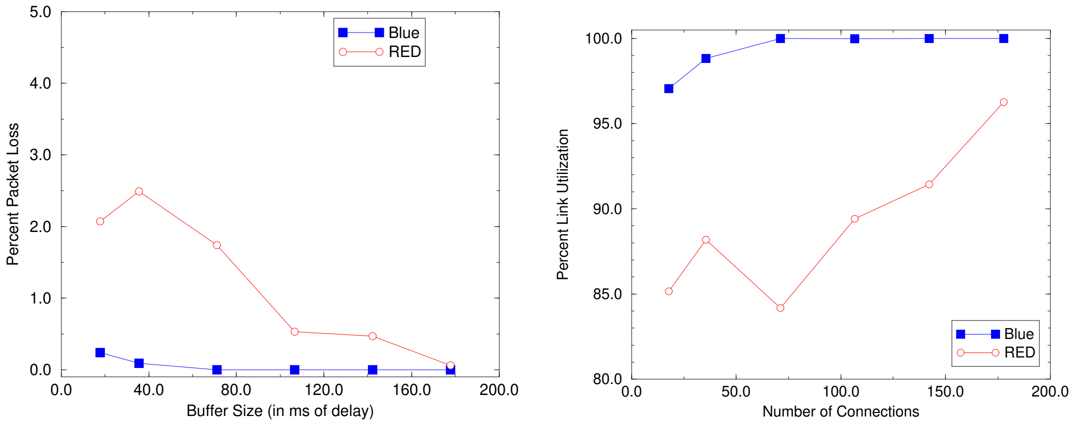
(a) p_m of BLUE

(b) $\frac{p_b}{1-count \times p_b}$ of RED

Figure 4.10: Marking behavior with ECN timeouts

Ethernet segments. In order to generate load on the system, a variable number of `netperf` [49] sessions are run from the *IBM PC 360* and the *Winbook XL* to the *IBM PC 365* and the *Thinkpad 770*. The router queue on the congested Ethernet interface of the *Intellistation Zpro* is sized at $50KB$ which corresponds to a queueing delay of about $40ms$. To ensure that the BLUE modifications did not create a bottleneck in the router, the testbed was reconfigured exclusively with Fast Ethernet segments and a number of experiments between network endpoints were run using the BLUE modifications on the intermediate routers. In all of the experiments, the sustained throughput was always above $80Mbs$.

Figures 4.13(a) and (b) show 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 while 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 five 10-second intervals and then averaged. Note that the TCP sources used in the experiment do not implement ECN timeouts. As Figure 4.13(a) shows, both the BLUE queue and the optimally configured RED queue maintain relatively high levels of throughput across all loads. However, since RED periodically allows the link to become underutilized, its throughput remains slightly below that of BLUE. As Figure 4.13(b) shows, RED sustains increasingly high packet loss as the number of connections is increased. Since aggregate TCP traffic becomes more aggressive as the number of connections increases, it becomes difficult for RED to maintain low loss rates. Fluctuations in queue lengths occur so abruptly that the RED algorithm



(a) Loss rates (b) Link utilization

Figure 4.11: Performance of RED and BLUE with ECN timeouts

oscillates between periods of sustained marking and packet loss to periods of minimal marking and link underutilization. In contrast, BLUE maintains relatively small packet loss rates across all loads. At higher loads, when packet loss is observed, BLUE maintains a marking probability which is approximately 1, causing it to mark every packet it forwards.

4.4 Stochastic Fair BLUE

Up until recently, the Internet has mainly relied on the cooperative nature of TCP congestion control in order to limit packet loss and fairly share network resources. Increasingly, however, new applications are being deployed which do not use TCP congestion control and are not responsive to the congestion signals given by the network. Such applications are potentially dangerous because they drive up the packet loss rates in the network and can eventually cause congestion collapse [34, 53]. In order to address the problem of non-responsive flows, a lot of work has been done to provide routers with mechanisms for protecting against them [16, 41]. The idea behind these approaches is to detect non-responsive flows and to limit their rates so that they do not impact the performance of responsive flows. This section describes and evaluates *Stochastic Fair* BLUE (SFB), a novel technique for protecting TCP flows against non-responsive flows using the BLUE algorithm. SFB is highly scalable and enforces fairness using an extremely small amount of state and a small amount of buffer space.

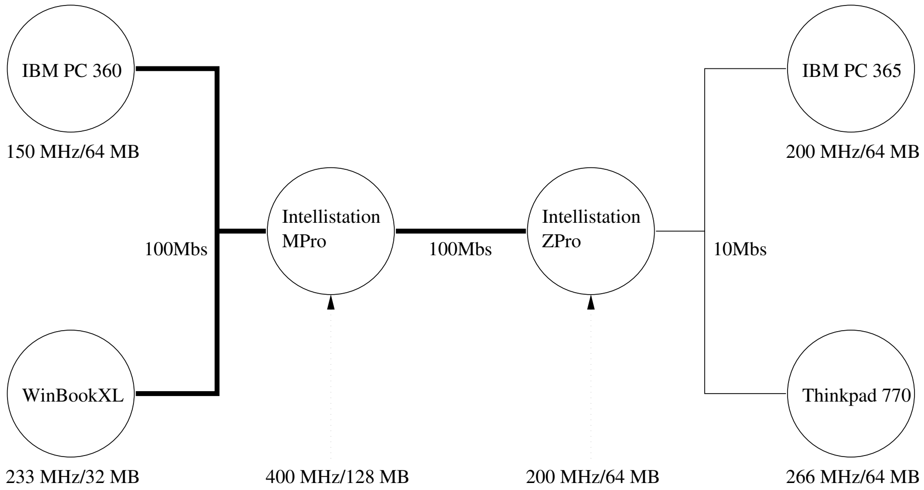
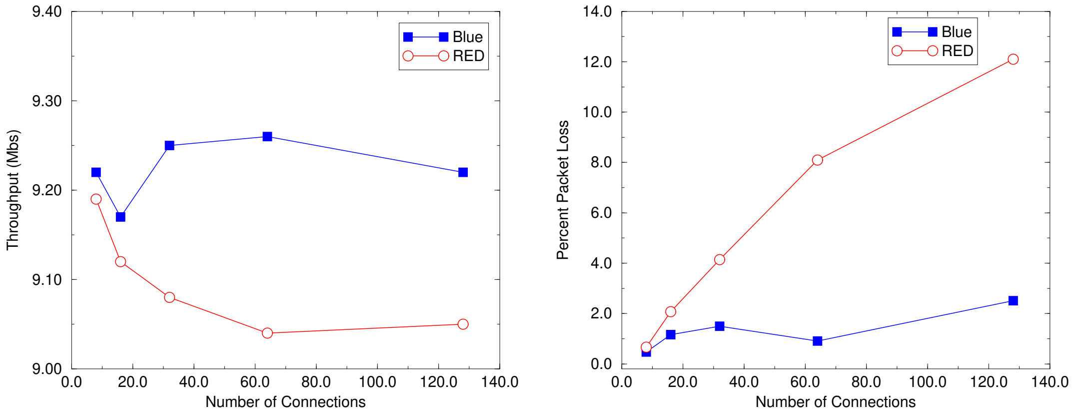


Figure 4.12: Experimental testbed

4.4.1 The algorithm

Figure 4.14 shows the basic SFB algorithm. SFB is a FIFO queueing algorithm that identifies and rate-limits non-responsive flows based on accounting mechanisms similar to those used with BLUE. SFB maintains $N \times L$ accounting bins. The bins are organized in L levels with N bins in each level. In addition, SFB maintains (L) independent hash functions, each associated with one level of the accounting bins. Each hash function maps a flow into one of the N accounting bins in that level. The accounting bins are used to keep track of queue occupancy statistics of packets belonging to a particular bin. This is in contrast to Stochastic Fair Queueing [47] (SFQ) where the hash function maps flows into separate queues. Each bin in SFB keeps a marking/dropping probability p_m as in BLUE, which is updated based on bin occupancy. As a packet arrives at the queue, it is hashed into one of the N bins in each of the L levels. If the number of packets mapped to a bin goes above a certain threshold (i.e., the size of the bin), p_m for the bin is increased. If the number of packets drops to zero, p_m is decreased.

The observation which drives SFB is that a non-responsive flow quickly drives p_m to 1 in all of the L bins it is hashed into. Responsive flows may share one or two bins with non-responsive flows, however, unless the number of non-responsive flows is extremely large compared to the number of bins, a responsive flow is likely to be hashed into at least one bin that is not polluted with non-responsive flows and thus has a normal p_m value. The decision to mark a packet is based on p_{min}, the minimum p_m value of all bins to which the flow is mapped into. If p_{min} is 1, the packet is



(a) Throughput

(b) Percent packet loss

Figure 4.13: Queue management performance

identified as belonging to a non-responsive flow and is then rate-limited. At this point, a number of options are available to limit the transmission rate of the flow. In this work, flows identified as being non-responsive are simply limited to a fixed amount of bandwidth. This policy is enforced by limiting the rate of packet enqueues for flows with p_{min} values of 1. Figure 4.15 shows an example of how SFB works. As the figure shows, a non-responsive flow drives up the marking probabilities of all of the bins it is mapped into. While the TCP flow shown in the figure may map into the same bin as the non-responsive flow at a particular level, it maps into normal bins at other levels. Because of this, the minimum marking probability of the TCP flow is below 1.0 and thus, it is not identified as being non-responsive. On the other hand, since the minimum marking probability of the non-responsive flow is 1.0, it is identified as being non-responsive and rate-limited.

Note that just as BLUE's marking probability can be used in SFB to provide protection against non-responsive flows, it is also possible to apply Adaptive RED's max_p parameter to do the same. In this case, a per-bin max_p value is kept and updated according to the behavior of flows which map into the bin. As with RED, however, there are two problems which make this approach ineffective. The first is the fact that a large amount of buffer space is required in order to get RED to perform well. The second is that the performance of a RED-based scheme is limited since even a moderate amount of congestion requires a max_p setting of 1. Thus, RED, used in this manner, has an extremely difficult time distinguishing between a non-responsive flow and moderate levels of congestion.

$B[l][n]$: L x N array of bins (L levels, N bins per level)

$enque()$

    Calculate hash function values h_0, h_1, ..., h_{L-1};

    Update bins at each level

    for $i = 0$ to $L - 1$

        if $(B[i][h_i].qlen > bin\_size)$

            $B[i][h_i].p_m$ += delta;

            Drop packet;

        else if ($B[i][h_i].qlen$ == 0)

            $B[i][h_i].p_m$ -= delta;

    p_{min} = min($B[0][h_0].p_m$ .. $B[L][h_L].p_m$);

    if (p_{min} == 1)

        $ratelimit()$

    else

        Mark/drop with probability p_{min};

Figure 4.14: SFB algorithm

4.4.2 Evaluation

Using ns, the SFB algorithm was simulated in the same network as in Figure 4.4 with transmission delays of $10ms$ on all links. The SFB queue is configured with $200KB$ of buffer space and maintains two hash functions each mapping to 23 bins. The size of each bin is set to 13, approximately 50% more than $\frac{1}{23}^{rd}$ of the available buffer space. Note that by allocating more than $\frac{1}{23}^{rd}$ the buffer space to each bin, SFB effectively “overbooks” the buffer in an attempt to improve statistical multiplexing. Notice that even with overbooking, the size of each bin is quite small. Since BLUE performs extremely well under constrained memory resources, SFB can still effectively maximize network efficiency. The queue is also configured to rate-limit non-responsive flows to $0.16Mbs$.

In the experiments, 400 TCP sources and one non-responsive, constant rate source are run for 100 seconds from randomly selected nodes in $(n_0, n_1, n_2, n_3, n_4)$ to randomly selected nodes in $(n_5, n_6, n_7, n_8, n_9)$. In one experiment, the non-responsive flow transmits at a rate of $2Mbs$ while

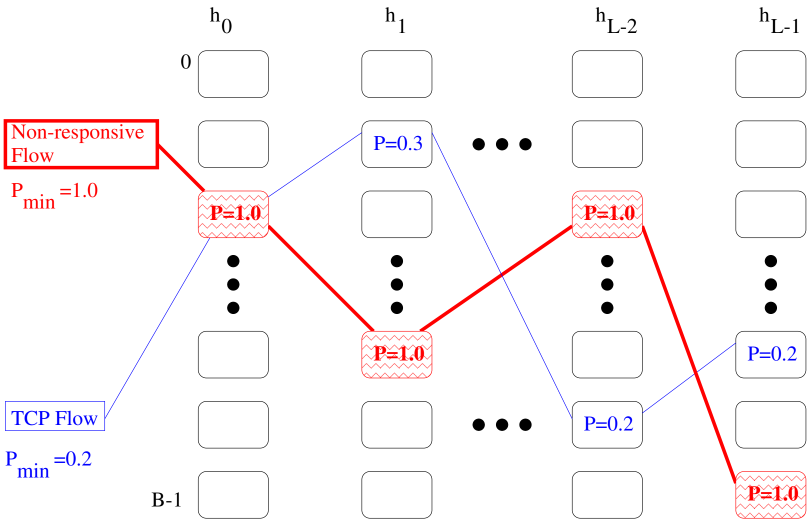


Figure 4.15: Example of SFB

in the other, it transmits at a rate of $45Mbs$. Table 4.1 shows the packet loss observed in both experiments for SFB. As the table shows, for both experiments, SFB performs extremely well. The non-responsive flow sees almost all of the packet loss as it is rate-limited to a fixed amount of the link bandwidth. In addition, the table shows that in both cases, a very small amount of packets from TCP flows are lost. Table 4.1 also shows the performance of RED. In contrast to SFB, RED allows the non-responsive flow to maintain a throughput relatively close to its original sending rate. As a result, the remaining TCP sources see a considerable amount of packet loss which causes their performance to deteriorate. Finally, the experiments were repeated using SFQ with an equivalent number of bins (i.e., 46 distinct queues) and a buffer more than twice the size $(414KB)$, making each queue equally sized at $9KB$. For each bin in the SFQ, the RED algorithm was applied with min_{th} and max_{th} values set at $2KB$ and $8KB$, respectively. As the table shows, SFQ with RED does an adequate job of protecting TCP flows from the non-responsive flow. However, in this case, partitioning the buffers into such small sizes causes a significant amount of packet loss to occur. Additional experiments show that as the amount of buffer space is decreased even further, the problem is exacerbated and the amount of packet loss increases considerably.

To qualitatively examine the impact that the non-responsive flow has on TCP performance, Figure 4.16 plots the throughput of all 400 TCP flows using SFB when the non-responsive flow sends at a $45Mbs$ rate. As the figure shows, SFB allows each TCP flow to maintain close to a fair share of the bottleneck link's bandwidth while the non-responsive flow is rate-limited to well below its

Packet Loss (Mbs)	$2Mbs$ non-responsive flow			$45Mbs$ non-responsive flow		
	SFB	RED	SFQ+RED	SFB	RED	SFQ+RED
Total	1.856	1.787	3.599	44.850	13.393	46.467
Non-responsive flow	1.846	0.034	1.034	44.841	10.324	43.940
All TCP flows	0.010	1.753	0.966	0.009	3.069	2.527

Table 4.1: SFB loss rates in Mbs (one non-responsive flow)

transmission rate. In contrast, Figure 4.17(a) shows the same experiment using normal RED queue management. The figure shows that the throughput of all TCP flows suffers considerably as the non-responsive flow is allowed to grab a large fraction of the bottleneck link bandwidth. Finally, Figure 4.17(b) shows that while SFQ with RED can effectively rate-limit the non-responsive flows, the partitioning of buffer space causes the fairness between flows to deteriorate as well. The large amount of packet loss induces a large number of retransmission timeouts across a subset of flows which causes significant amounts of unfairness [48]. Thus, through the course of the experiment, a few TCP flows are able to grab a disproportionate amount of the bandwidth while many of the flows receive significantly less than a fair share of the bandwidth across the link. In addition to this, SFQ with RED allows $\frac{1}{46}^{th}$ of the 400 flows to be mapped into the same queue as the non-responsive flow. Flows that are unlucky enough to map into this bin receive an extremely small amount of the link bandwidth. SFB, in contrast, is able to protect all of the TCP flows in this experiment.

4.4.3 Limitations of SFB

While it is clear that the basic SFB algorithm can protect TCP-friendly flows from non-responsive flows without maintaining per-flow state, it is important to understand how it works and its limitations. SFB effectively uses L levels with N bins in each level to create N^L virtual buckets. This allows SFB to effectively identify a single non-responsive flow in an N^L flow aggregate using $O(L * N)$ amount of state. For example, in the previous section, using two levels with 23 bins per level effectively creates 529 buckets. Since there are only 400 flows in the experiment, SFB is able to accurately identify and rate-limit a single non-responsive flow without impacting the performance of any of the individual TCP flows. As the number of non-responsive flows increases, the number

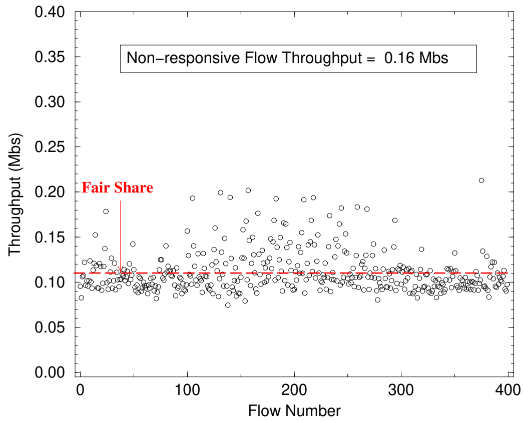


Figure 4.16: Bandwidth of TCP flows using SFB $(45Mbs$ flow)

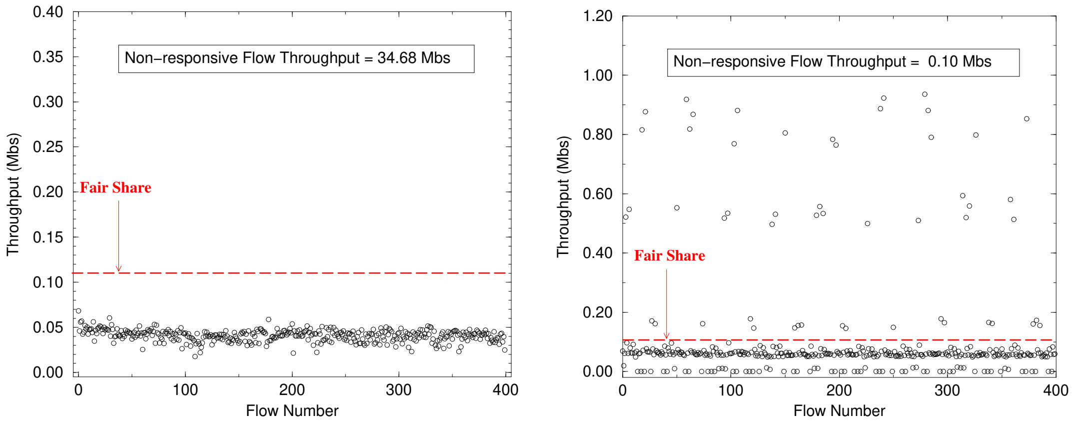
of bins which become "polluted" or have p_m values of 1 increases. Consequently, the probability that a responsive flow gets hashed into bins which are all polluted, and thus becomes misclassified, increases. Clearly, misclassification limits the ability of SFB to protect well behaved TCP flows.

Using simple probabilistic analysis, Equation (4.1) gives a closed-form expression of the probability that a well-behaved TCP flow gets misclassified as being non-responsive as a function of number of levels (L), the number of bins per level (B), and the number of non-responsive/malicious flows (M), respectively.

$$p = [1 - (1 - \frac{1}{B})^M]^L \tag{4.1}$$

In this expression, when L is 1, SFB behaves much like SFQ. The key difference is that SFB using one level is still a FIFO queueing discipline with a shared buffer while SFQ has separate per-bin queues and partitions the available buffer space amongst them.

Using the result from Equation (4.1), it is possible to optimize the performance of SFB given *a priori* information about its operating environment. Suppose the number of simultaneously active non-responsive flows can be estimated (M) and the amount of memory available for use in the SFB algorithm is fixed (C). Then, by minimizing the probability function in Equation (4.1) with the additional boundary condition that $L \times N = C$, SFB can be tuned for optimal performance. To demonstrate this, the probability for misclassification across a variety of settings is evaluated. Figure 4.18(a) shows the probability of misclassifying a flow when the total number of bins is fixed at 90. Figure 4.18(b) shows the same probability function when the total number of bins is fixed at 900. In these figures, the number of levels used in SFB along with the number of non-responsive

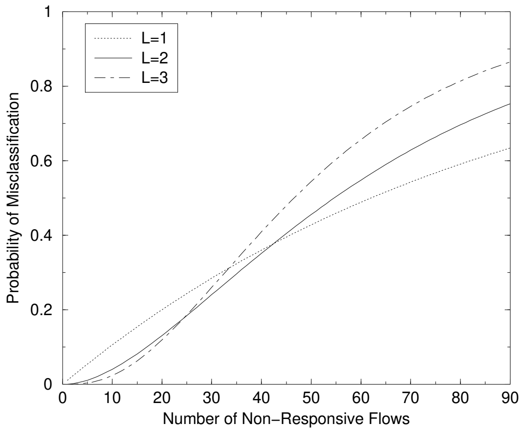


(a) RED (b) SFQ+RED

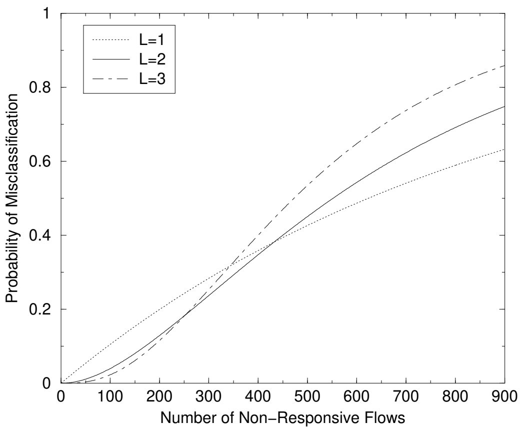
Figure 4.17: Bandwidth of TCP flows using RED and SFQ ($45Mbs$ flow)

flows are varied. As the figures show, when the number of non-responsive flows is small compared to the number of bins, the use of multiple levels keeps the probability of misclassification extremely low. However, as the number of non-responsive flows increases past half the number of bins present, the single level SFB queue affords the smallest probability of misclassification. This is due to the fact that when the bins are distributed across multiple levels, each non-responsive flow pollutes a larger number of bins. For example, using a single level SFB queue with 90 bins, a single non-responsive flow pollutes only one bin. Using a two-level SFB queue with each level containing 45 bins, the number of effective bins is 45×45 (2025). However, a single non-responsive flow pollutes two bins (one per level). Thus, the advantage gained by the two-level SFB queue is lost when additional non-responsive flows are added, as a larger fraction of bins become polluted compared to the single-level situation.

In order to evaluate the performance degradation of SFB as the number of non-responsive flows increases, Figure 4.19 shows the bandwidth plot of the 400 TCP flows when 4 and 8 non-responsive flows are present. In these experiments, each non-responsive flow transmits at a rate of $5Mbs$. As Equation (4.1) predicts, in an SFB configuration that contains two levels of 23 bins, 2.65% of the TCP flows (11) are misclassified when 4 non-responsive flows are present. Similarly, when 8 non-responsive flows are present, 8.96% (36) of the TCP flows are misclassified. When the number of non-responsive flows approaches N, the performance of SFB deteriorates quickly as an increasing number of bins at each level becomes polluted. In the case of 8 non-responsive flows, approxi-



(a) 90 bins



(b) 900 bins

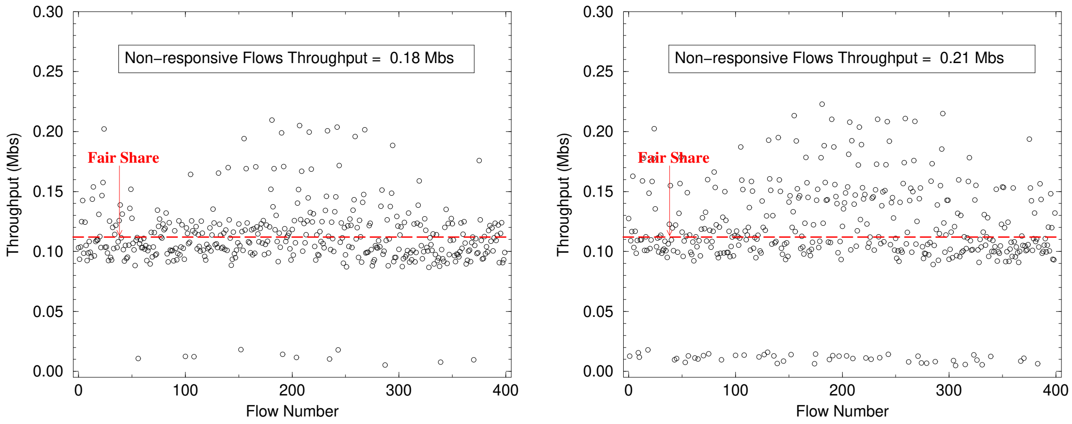
Figure 4.18: Probability of misclassification

mately 6 bins or one-fourth of the bins in each level are polluted. As the figure shows, the number of misclassified flows matches the model quite closely. Note that even though a larger number of flows are misclassified as the number of non-responsive flows increases, the probability of misclassification in a two-level SFB still remains below that of SFQ or a single-level SFB. Using the same number of bins (46), the equation predicts that SFQ and a single-level SFB misclassify 8.42% of the TCP flows (34) when 4 non-responsive flows are present and 16.12% of the TCP flows (64) when 8 non-responsive are present.

4.4.4 SFB with moving hash functions

In this section, two basic problems with the SFB algorithm are addressed. The first, as described above, is to mitigate the effects of misclassification. The second is to be able to detect when non-responsive flows become responsive and to reclassify them when they do.

The idea behind SFB with moving hash functions is to periodically or randomly reset the bins and change the hash functions. A non-responsive flow will continually be identified and rate-limited regardless of the hash function used. However, by changing the hash function, responsive TCP flows that happen to map into polluted bins will potentially be remapped into at least one unpolluted bin. In many ways the effect of using moving hash functions is analogous to channel hopping in CDMA [33, 64] systems. It essentially reduces the likelihood of a responsive connection being continually penalized due to erroneous assignment into polluted bins.



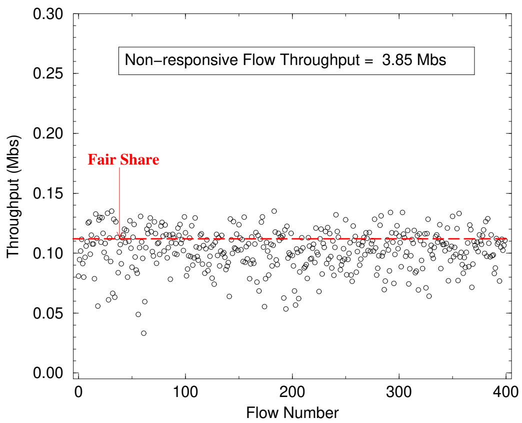
(a) 4 non-responsive flows

(b) 8 non-responsive flows

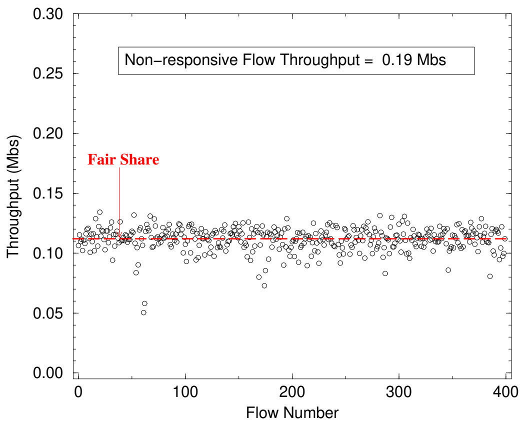
Figure 4.19: Bandwidth of TCP flows using SFB

To show the effectiveness of this approach, the idea of moving hash functions was applied to the experiment in Figure 4.19(b). In this experiment, 8 non-responsive flows along with 400 responsive flows share the bottleneck link. To protect against continual misclassification, the hash function is changed every two seconds. Figure 4.20(a) shows the bandwidth plot of the experiment. As the figure shows, SFB performs fairly well. While flows are sometimes misclassified causing a degradation in performance, none of the TCP-friendly flows are shut out due to misclassification. This is in contrast to Figure 4.19 where a significant number of TCP flows receive very little bandwidth.

While the moving hash functions improve fairness across flows in the experiment, it is interesting to note that every time the hash function is changed and the bins are reset, non-responsive flows are temporarily placed on “parole”. That is, non-responsive flows are given the benefit of the doubt and are no longer rate-limited. Only after these flows cause sustained packet loss, are they identified and rate-limited again. Unfortunately, this can potentially allow such flows to grab much more than their fair share of bandwidth over time. For example, as Figure 4.20(a) shows, non-responsive flows are allowed to consume $3.85Mbs$ of the bottleneck link. One way to solve this problem is to use two sets of bins. As one set of bins is being used for queue management, a other set of bins using the next set of hash functions can be warmed up. In this case, any time a flow is classified as non-responsive, it is hashed using the second set of hash functions and the marking probabilities of the corresponding bins in the warmup set are updated. When the hash functions are switched, the bins which have been warmed up are then used. Consequently, non-responsive flows are rate-limited



(a) Moving hash



(b) Double buffered moving hash

Figure 4.20: Bandwidth of TCP flows using modified SFB algorithms

right from the beginning. Figure 4.20(b) shows the performance of this approach. As the figure shows, the double buffered moving hash effectively controls the bandwidth of the non-responsive flows and affords the TCP flows a very high level of protection.

One of the advantages of the moving hash function is that it can quickly react to non-responsive flows which become TCP-friendly. In this case, changing the hash bins places the newly reformed flow out on parole for good behavior. Only after the flow resumes transmitting at a high rate, is it again rate-limited. To show this, an additional experiment was run using the same experimental setup as above. In this experiment, one non-responsive flow with a transmission rate of $5Mbs$ and one oscillating flow is run between network endpoints. The oscillating flow transmits at $5Mbs$ from $t = 10s$ to $t = 30s$ and from $t = 50s$ to $t = 70s$. At all other times, the flow transmits at $0.10Mbs$, approximately a fair share of the bottleneck link. Table 4.2 shows the packet loss rates in the experiment. As the table shows, the first non-responsive flow sees a sustained packet loss rate throughout the experiment which effectively limits its throughput to well below its transmission rate. The table also shows that when the second flow transmits at $5Mbs$, it observes a sustained packet loss rate as a large fraction of its packets are dropped by the queue. When the second flow cuts its transmission rate to a fair share of the link's bandwidth, it is reclassified and a very small fraction of its packets are dropped. Finally, the table shows that all 400 TCP flows see a minimal amount of packet loss throughout the experiment. Figure 4.21 shows the bandwidth plot for the TCP flows in the experiment. As shown in the figure, SFB protects the TCP flows from the non-responsive

	Loss Rates (in Mbs)			
	$10s$-$30s$	$30s$-$50s$	$50s$-$70s$	$70s$-$100s$
Non-responsive Flow	4.866	4.849	4.898	4.863
Oscillating Flow	4.871	0.025	4.845	0.017
TCP Flows	0.402	0.358	0.260	0.324
Total	10.139	5.232	10.003	5.204

Table 4.2: SFB loss rates (one non-responsive, one oscillating flow)

flows, thus allowing them to maintain close to a fair share of the bottleneck link.

4.4.5 Round-trip time sensitivity

The previous experiments with SFB use a network topology in which all of the connections have approximately the same round-trip time. When a large number of connections with varying round-trip times are used with SFB, fairness between flows can deteriorate. It has been shown that TCP connections with smaller round-trip times can dominate the bandwidth on the bottleneck link since their window increases are clocked more frequently. When a small number of such connections are present, SFB can mitigate this problem somewhat. Similar to the non-responsive flow cases above, TCP connections with small round-trips slowly drive the marking probability of their bins higher. Thus, when p_{min} is calculated, they receive a larger fraction of congestion notification. However, when a large number of TCP flows with varying round-trip times are present, this mechanism breaks down just as SFB breaks down with a large number of non-responsive flows.

Figure 4.22 shows the performance of RED and SFB using the network shown in Figure 4.4. Using this network, 400 sources are randomly started between network end points. As the figure shows, both RED and SFB show biases towards connections with smaller round-trip times. However, since all of the flows still use TCP, the amount of unfairness between flows is limited.

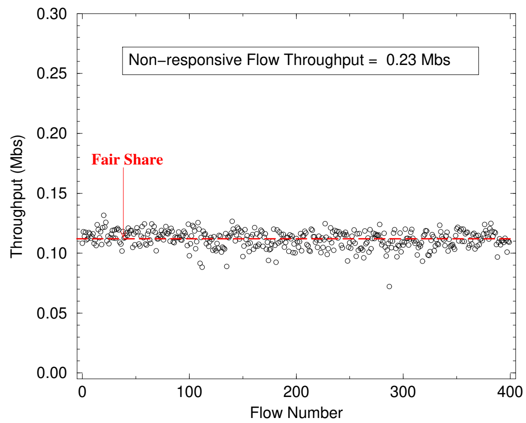


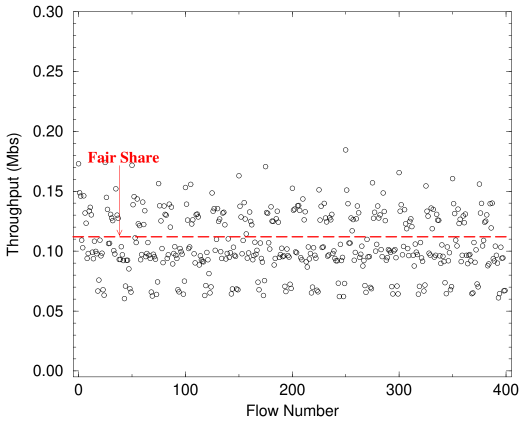
Figure 4.21: Bandwidth of TCP flows (one non-responsive, one oscillating flow)

4.5 Comparisons to Other Approaches

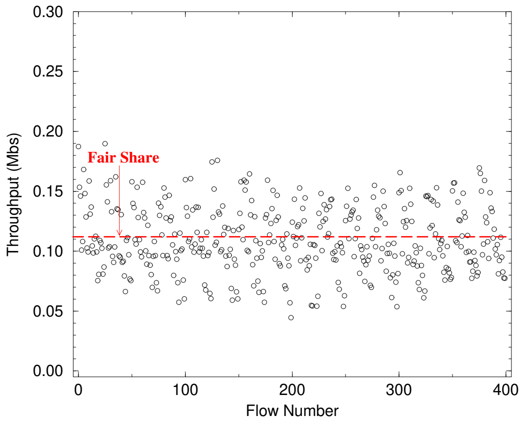
SFB provides one particular solution for identifying and rate-limiting non-responsive flows, thereby enforcing fairness. This section compares SFB to other related approaches.

4.5.1 RED with penalty box

The RED with penalty box approach takes advantage of the fact that high bandwidth flows see proportionally larger amounts of packet loss. By keeping a finite log of recent packet loss events, this algorithm identifies flows which are non-responsive based on the log [46]. Flows which are identified as being non-responsive are then rate-limited using a mechanism such as class-based queueing [27]. While this approach may be viable under certain circumstances, it is unclear how the algorithm performs in the face of a large number of non-responsive flows. Unless the packet loss log is large, a single set of high bandwidth flows can potentially dominate the loss log and allow other, non-responsive flows to go through without rate-limitation. In addition, flows which are classified as non-responsive remain in the "penalty box" even if they subsequently become responsive to congestion. A periodic and explicit check is thus required to move flows out of the penalty box. Finally, the algorithm relies on a TCP-friendliness check in order to determine whether or not a flow is non-responsive. Without *a priori* knowledge of the round-trip time of every flow being multiplexed across the link, it is difficult to accurately determine whether or not a connection is TCP-friendly.



(a) RED



(b) SFB

Figure 4.22: Bandwidth of TCP flows over varying round-trip times.

4.5.2 FRED

Another proposal for using RED mechanisms to provide fairness is Flow-RED (FRED) [41]. The idea behind FRED is to keep state based on instantaneous queue occupancy of a given flow. If a flow continually occupies a large amount of the queue's buffer space, it is detected and limited to a smaller amount of the buffer space. While this scheme provides rough fairness in many situations, since the algorithm only keeps state for flows which have packets queued at the bottleneck link, it requires a large amount of buffers to work well. Without sufficient buffer space, it becomes hard for FRED to detect non-responsive flows since they may not have enough packets continually queued to trigger the detection mechanism. In addition, non-responsive flows are immediately re-classified as being responsive as soon as they clear their packets from the congested queue. For small queue sizes, it is quite easy to construct a transmission pattern which exploits this property of FRED in order to circumvent its protection mechanisms. Note that SFB does not directly rely on queue occupancy statistics, but rather long-term packet loss and link utilization behavior. Because of this, SFB is better suited for protecting TCP flows against non-responsive flows using a minimal amount of buffer space. Finally, as with the packet loss log approach, FRED also has a problem when dealing with a large number of non-responsive flows. In this situation, the ability to distinguish these flows from normal TCP flows deteriorates considerably since the queue occupancy statistics used in the algorithm become polluted. By not using packet loss as a means for identifying non-responsive flows, FRED cannot make the distinction between N TCP flows multiplexed across a link versus N

non-responsive flows multiplexed across a link.

4.5.3 RED with per-flow queueing

A RED-based, per-active flow approach has been proposed for providing fairness between flows [63]. The idea behind this approach is to do per-flow accounting and queueing only for flows which are active. The approach argues that since keeping a large amount of state is feasible, per-flow queueing and accounting is possible even in the core of the network. The drawbacks of this approach is that it provides no savings in the amount of state required. If N flows are active, $O(N)$ amount of state must be kept to isolate the flows from each other. In addition, this approach does not address the large amount of legacy hardware which exists in the network. For such hardware, it may be infeasible to provide per-flow queueing and accounting. Because SFB provides considerable savings in the amount of state and buffers required, it is a viable alternative for providing fairness efficiently.

4.5.4 Stochastic Fair Queueing

Stochastic Fair Queueing (SFQ) is similar to an SFB queue with only one level of bins. The biggest difference is that instead of having separate queues, SFB uses the hash function for accounting purposes. Thus, SFB has two fundamental advantages over SFQ. The first is that it can make better use of its buffers. SFB gets some statistical multiplexing of buffer space as it is possible for the algorithm to overbook buffer space to individual bins in order to keep the buffer space fully utilized. As described in Section 4.4.2, partitioning the available buffer space adversely impacts the packet loss rates and the fairness amongst TCP flows. The other key advantage is that SFB is a FIFO queueing discipline. As a result, it is possible to change the hash function on the fly without having to worry about packet re-ordering caused by mapping of flows into a different set of bins. Without additional tagging and book-keeping, applying the moving hash functions to SFQ can cause significant packet re-ordering.

4.5.5 Core-Stateless Fair Queueing

Core-Stateless Fair Queueing [62] (CSFQ) is a highly scalable approach for enforcing fairness between flows without keeping any state in the core of the network. The approach relies on per-flow accounting and marking at the edge of the network in conjunction with a probabilistic dropping mechanism in the core of the network. The idea behind CSFQ is to estimate the rate of the flow at the ingress of the network or network cloud and to attach an estimate of the flow's sending rate to *every* packet that the flow sends. Given this label, intermediate routers at congested links in the network calculate a dropping probability which is derived from an estimate of a fair share of the bottleneck link capacity and the rate of the flow as identified in the label.

While CSFQ provides an elegant and efficient solution to providing fairness, it relies on the use of additional information that is carried in every packet of the flow. Thus, the scheme trades off overhead in the packet header at every network link for resource management overhead at the bottleneck router. In addition, it requires that both intermediate routers and edge devices adhere to the same labeling and dropping algorithm. A misconfigured or poorly implemented edge device can significantly impact the fairness of the scheme. SFB, on the other hand, does not rely on coordination between intermediate routers and edge markers and can peform well without placing additional overhead in packet headers.

4.6 Conclusion and Future Work

This chapter has demonstrated the inherent weakness of current active queue management algorithms which use queue occupancy in their algorithms. In order to address this problem, a fundamentally different queue management algorithm called BLUE has been designed and evaluated. BLUE uses the packet loss and link utilization history of the congested queue, instead of queue lengths, to manage congestion. In addition to BLUE, this chapter has proposed and evaluated SFB, a novel algorithm for scalably and accurately enforcing fairness amongst flows in a large aggregate. Using SFB, non-responsive flows can be identified and rate-limited using a very small amount of state.

As part of on-going work, several extensions to BLUE and SFB are being considered. In order to improve the adaptiveness and accuracy of BLUE, a range of real, Internet, traffic traces is being studied. By understanding how quickly and how often traffic changes across bottleneck links, the BLUE algorithm and its parameters can be optimally parameterized. To improve the effectiveness of SFB, additional mechanisms for managing non-responsive flows are being examined. In this chapter, non-responsive flows were rate-limited to a fixed amount of bandwidth across the bottleneck link. However, it is possible to rate-limit non-responsive flows to a fair share of the link's capacity. One way to do this is to estimate both the number of non-responsive flows and the total number of flows going through the bottleneck. Using this information, the rate-limiting mechanism can be set accordingly. Another possible mechanism to find the number of "polluted" bins and use it to derive the fraction of flows which are non-responsive. Assuming perfect hash functions, this can be directly derived from simple analytical models of SFB as described in Section 4.4. Finally, the development of an "enhanced" BLUE queue management algorithm which is similar to "enhanced" RED [19, 20] is being considered. By using BLUE, the buffer requirements needed to support differentiated services can be greatly reduced.

CHAPTER 5

UNDERSTANDING TCP DYNAMICS IN A DIFFERENTIATED SERVICES INTERNET

5.1 Introduction

The previous two chapters have addressed how to improve the efficiency of delivering best-effort service in today's Internet. While this is important to the future success of the Internet, another important problem facing the Internet today is supporting additional services for emerging applications. As described in Chapter 2, the IETF is considering a more evolutionary approach to provide service differentiation in the Internet using the type-of-service (ToS) bits in the IP header. Through the Differentiated Services (DIFFSERV) working group, a small set of building blocks are being defined which allow routers to scalably provide service differentiation. While it is relatively clear how to build predictable services using the protocols and mechanisms provided by RSVP and INTSERV, the ability to construct predictable services using the coarse-grained mechanisms provided by DIFFSERV is an open issue.

This chapter presents an implementation of a controlled-load service variant using the simple priority mechanisms provided by DIFFSERV, and in particular, the AF PHB. Controlled-load service [68] is one of the services which has been standardized by the INTSERV working group for deployment in the network. One of the salient features of this service is that it provides predictable, end-to-end bandwidth assurances to applications. However, because of the implicit need to process and forward packets from sources on a per-flow basis, the controlled-load service, along with many of the other INTSERV services has not seen widespread deployment in the Internet due to the amount of overhead involved in supporting it.

By eliminating the per-flow forwarding aspects associated with INTSERV-style services, this chapter describes a more scalable implementation of controlled-load service. This implementation uses a simple extension to the queueing mechanisms in today's routers coupled with modifications to TCP's congestion control mechanisms. These modifications enable the network to guarantee a minimal level of end-to-end throughput to different network sessions. In addition, any residual network capacity is shared in a socially cooperative fashion, in a manner similar to the one in use in the Internet today by applications using TCP. In this scheme, each reserved session is associated with a traffic envelope. Traffic is policed at the source and packets conforming to the envelope are marked (with the AF PHB). Non-conformant traffic and best-effort traffic is injected into the network unmarked. At the routers an enhanced RED (ERED) [26] algorithm is used. In ERED, both marked and unmarked packets share the same FIFO queue. When the queue length at the router exceeds a certain threshold, packets are dropped randomly as done in RED gateways. However, unlike standard RED gateways where all packets have the same drop probability, in the enhanced RED (ERED) gateway, marked packets have a lower drop probability than the unmarked packets.

The service realized by the mechanism described above is an interpretation of the controlled-load service. By definition, traffic belonging to a controlled-load session and conforming to the associated traffic envelope sees very little loss and very little queueing delay through the network. Non-conformant controlled-load traffic is treated as best-effort traffic. By using a common queue for best-effort and conformant controlled-load traffic, the recommended delay targets for conformant controlled-load traffic is relaxed. This laxity not only simplifies the implementation and reduces packet handling overheads at the routers, but also helps maintain packet ordering. Note that ERED only ensures a low loss rate to conformant controlled-load traffic. Because many elastic and tolerant playback applications can withstand a reasonable amount of queueing delay, the use of a shared FIFO to service both marked and unmarked packets is feasible.

There are many ways to potentially implement controlled-load service using a variety of transport protocols other than TCP and a variety of queueing mechanisms other than ERED. While other approaches may perform equally well, TCP is the only one considered since its use is ubiquitous. In addition, ERED queueing is used because (1) it does not require per-flow queueing which may not scale well and (2) it maintains FIFO ordering of packets which is important to TCP's congestion control mechanisms.

Although controlled-load service can be used in conjunction with any transport protocol, the focus in this chapter is on TCP. TCP is examined since (1) an overwhelming number of applications use TCP as the transport protocol of choice, and (2) TCP has a well-developed congestion and flow control mechanism that makes it an interesting case study. While some of the tolerant playback applications may not use TCP, the mechanisms described here can easily be applied to other transport protocols, such as RTP. The objective of this chapter is to understand and to improve the end-to-end control mechanisms used in TCP in a network which supports both best-effort and priority-based mechanisms. The analysis of the behavior of unmodified TCP over a network which does priority marking has particular relevance for current DIFFSERV proposals based on AF.

5.2 Integrated Services

The RSVP and the INTSERV working groups in the IETF have defined several protocols and standards to support controlled-load and other integrated services in the Internet. This section reviews these standards and shows how the proposed enhancements fit into the IETF-defined framework.

5.2.1 Policing and marking

To avail itself of a reservation, a connection has to specify a traffic envelope, called *Tspec*. The Tspec includes a long-term average rate (r_m), a short-term peak rate (r_p), and the maximum size (b) of a burst of data generated by the application. For example, for an application generating MPEG-encoded video, the average rate could be the long-term data rate, the peak rate could be the link bandwidth at the source, and the burst size could be the maximum size of a frame. The Tspec also specifies the maximum and minimum packet sizes to be used by the application. Connections are monitored and policed at the network entry points. This could be either at the source, or at the boundary between the corporate or campus intranet and the Internet. Packet classification and service differentiation also takes place at the routers. The service priority given to a packet is a function of the Tspec, and in the case of some service classes, a separate service specification known as *Rspec*. For controlled-load service, no Rspec is specified.

In order to police traffic at the source, token buckets are used [56]. The token generation process follows the Tspec advertised by the source. That is, the long-term average rate of token generation is t_m, the short-term peak rate of token generation is t_p, and the depth of the token bucket is b. Each time a packet is injected into the network, if sufficient tokens are available, an equivalent number of tokens are considered consumed. If tokens are not present at the time of transmission, the packet is treated as non-conformant. In the presence of an AF-style marking facility, classification is only required at the network entry point and not at interior routers. Conformant controlled-load traffic is AF-marked at network entry points before being injected into the network while non-conformant controlled-load traffic and best-effort traffic is injected into the network unmarked. In the absence of a marking facility, IP datagrams have to be passed through a classifier at the source, as well as at the routers, to determine which flows they belong to and to determine whether they are in violation of, or in conformance with, the advertised Tspecs of the flows. In the rest of the chapter, it is assumed that AF-marking is available in the network.

5.2.2 Packet handling

The routers perform admission control for controlled-load connections. Admission control algorithms are not discussed in this chapter, but, for the purpose of the experiments, it is assumed that the aggregate reservation levels at the routers are within their capacities. In addition to performing admission control, the routers also need to support service differentiation between marked (conformant controlled-load) and unmarked (non-conformant controlled-load and best-effort) packets. One obvious approach to providing different services to marked and unmarked packets is to maintain separate queues for each class and serving them according to their scheduling priority. Another approach is to use a common FIFO queue for both compliant and non-compliant traffic. A common FIFO queue not only simplifies the scheduling functionality at the router, it also helps maintain packet ordering in controlled-load connections. Although maintaining packet ordering is not a requirement, failure to do so may have serious performance impacts on transport protocols such as TCP.

In order to provide service differentiation between marked and unmarked packets, a selective packet discard mechanism based on an enhanced version of the RED is used. Enhanced Random Early Detection (ERED) is a minor modification to the original RED algorithm. In ERED, the thresholds only apply to unmarked packets. Unmarked packets are randomly dropped when the average queue length exceeds $min_{th}(unmark)$ and are all dropped when the average queue length exceeds $max_{th}(unmark)$. Marked packets are only dropped when the queue is full. In order to ensure low loss of marked packets, min_{th} and max_{th} values need to be set appropriately. For example, in a system with n controlled-load sessions with peak rates of r_p^i, $i = 1, 2, \ldots, n$, a service rate of L, and a buffer of length B, the thresholds must be set so that they can roughly ensure that no marked packets are dropped[1]. In particular, the following equation should hold:

$$\left(\sum_{i=1}^{n} r_p^i - L\right) \times \frac{max_{th}(unmark)}{L} < B - max_{th}(unmark)$$

Note that the maximum number of unmarked packets that can be in the queue at any time is around (max_{th}(unmark)). It takes at most $\frac{max_{th}(unmark)}{L}$ to completely drain the queue of unmarked packets. Given the maximum aggregate arrival rate of marked packets $\sum r_p^i$ and the service rate L, the rate of increase of the queue occupancy is $\sum r_p^i - L$ since transmission of an unmarked packet makes room for an incoming marked packet. Hence, the amount of excess buffer space needed to ensure no marked packets are dropped is the product of ($\sum r_p^i - L$), the rate of increase in queue occupancy and $\frac{max_{th}(unmark)}{L}$, the time needed to drain the unmarked packets. Unfortunately, since the thresholds such as max_{th} are triggered by an average queue length calculation, these settings can still lead to unnecessary losses in compliant packets. For example, additional queueing work has shown the use of absolute thresholds for limiting unmarked packets can effectively prevent loss of marked packets [10, 30]. While ERED works well for the experiments in this study, a queue management algorithm which reserves a portion of the queue for marked packets while performing RED on the remaining portion might be ideal for controlling congestion while effectively supporting the priority marking.

An appropriately parameterized ERED queue can still be used to guarantee low loss rate to conformant controlled-load traffic. Since it uses a common FIFO queue, though, the delay experienced by the conformant controlled-load traffic and best-effort traffic is the same. It is possible to parameterize ERED queues to control the queue size, and hence, the queueing delay. However, a small queue size may lead to high loss rates for unmarked packets. An alternative approach is to maintain separate queues for controlled-load and best-effort traffic. Separation of traffic classes is likely to improve the delay performance of controlled-load traffic. However, it complicates bandwidth

[1] It is assumed that the duration of bursts is the same for all sources. This assumption can be relaxed for more precise admission control.

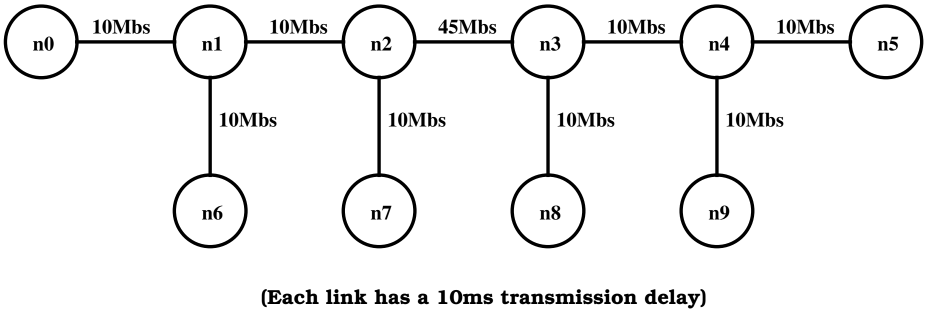


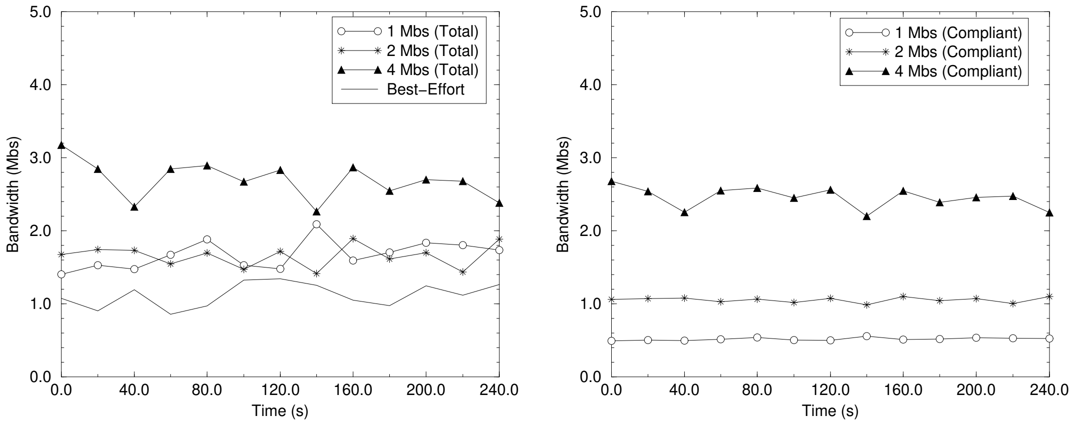
Figure 5.1: Network topology

sharing between non-conformant controlled-load and best-effort traffic. The router would have to use weighted fair queueing to ensure equal-fair share of excess bandwidth to non-conformant controlled-load and best-effort traffic. Consequently, it has to monitor the number of active connections in the controlled-load and best-effort classes and has to adjust the weights dynamically depending on the number of connections in each class.

Finally, there are many other ways of realizing controlled-load service. Some of these mechanisms, such as class-based queueing [27] and weighted fair queueing [5, 12, 14, 28, 29, 57, 60], can be used to accurately implement controlled-load and other service classes defined by the INTSERV working group. Section 5.8 describes how ERED and services based on priority handling of packets can be embedded in a more fully evolved integrated and differentiated services Internet. In particular, the ability to effectively embed ERED in a class-based queueing framework is shown.

5.3 Understanding TCP Dynamics

This section is devoted to the study of TCP dynamics in a differentiated services environment. For the purpose of the experiments, the `ns` simulator was modified [46]. For most of the experiments reported here, the NewReno variant of TCP [32] is used. The simulator was then modified by adding policing and extending the RED queueing discipline. For the experiments in this section, a simple network topology shown in Figure 5.1 is considered. The capacity of each bi-directional link is labeled and has a transmission delay of $10ms$. Connections requesting a reservation specify a peak and a mean rate of service, and the maximum size of a burst. At the source, tokens are generated at the service rate and are accumulated in a token bucket. The depth of the token bucket



(a) Total throughput (b) Compliant throughput.

$$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$$
$$BucketDepth = 50ms$$

Figure 5.2: Effect of reservation on end-to-end throughput.

is the same as the maximum burst size specified by the source. Throughout this chapter, the token bucket size is measured in units of time. In units of tokens, it is equivalent to token generation rate multiplied by the bucket size in units of time. The peak rate is set to the link speed by default. TCP segments belonging to the reserved connections are transmitted as marked datagrams if there are sufficient tokens available in the token bucket at the time of transmission. Otherwise, they are sent as unmarked datagrams. TCP segments belonging to best-effort connections are sent as unmarked datagrams. It is assumed that sources are greedy, that is, they always have data to send.

5.3.1 Effect of service rate

In order to investigate the effect of service rate on end-to-end throughput, three connections with reservations of $1Mbs$, $2Mbs$, and $4Mbs$, and three best-effort connections from node $n0$ to $n5$ were run. Each controlled-load source used a token bucket of depth $50ms$ and each node has a $100KB$ ERED queue with max_{th} of $80KB$ and min_{th} of $20KB$. The maximum drop probability of the unmarked packets for this experiment was 0.02. This probability is chosen in order to make early detection aggressive enough to control the length of the queue. Note that a drop probability which is too small makes the early detection mechanism ineffective while a drop probability which is too large can lead to underutilization of the link as described in Chapter 3.

Figure 5.2(a) shows the throughput seen by each connection. Throughput is computed by measuring the data received at the receiver over an observation period and dividing it by the observation interval. Figure 5.2(b) plots the compliant throughput seen by connections with reservations. This is the portion of the throughput that is contributed by marked packets. Ideally, it should be equal to the reserved rate of service. From Figure 5.2(a), it is evident that connections with higher reservations generally see better throughput than connections with lower or no reservations. However, as shown in Figure 5.2(b), the compliant portions of the bandwidth received by all reserved connections are less than their respective service rates.

The explanation for the observations from Figure 5.2 lies in the flow and congestion control mechanisms used by TCP. The TCP sessions with reservations exercise their flow and congestion control mechanisms in the same way as best-effort connections. However, they have a lower probability of losing a packet at the routers since their marked packets have lower (in this case close to zero) probability of getting dropped. Because connections with higher reservations mark their packets at a higher rate, they have a decreased probability of having a packet dropped. This is why connections with higher reservations see higher throughput than connections with lower or no reservations. However, as observed from Figure 5.2(b), TCP fails to fully exploit the benefits of the reservation. The compliant part of the throughput is less than the reservation levels in all cases. Since marked packets are not dropped by the network it is apparent that the source is not generating a sufficient number of marked packets to keep the reserved pipe full. Since the sender is a greedy source, it is the TCP congestion control mechanism that is responsible for throttling the source. The tokens, however, are generated at a rate commensurate with the reservation. If the source does not have enough or is unable to transmit packets, the token bucket fills up and ultimately overflows causing token loss.

Figure 5.3 shows the packet trace of the connection with $4Mbs$ reservation over a five-second interval. The plot shows the sequence number (modulo 200)[2], the congestion window of the sender, and the number of lost tokens (given in packets modulo 200) for the connection. A positive slope of the lost token curve indicates a non-zero token loss rate throughout the observation period. The windowing mechanism used by TCP is partly responsible for this phenomenon.

TCP uses two windows for the purpose of flow and congestion control. The receiver maintains

[2]Segments of size $1KB$ are used. The sequence number is the sender's packet sequence number.

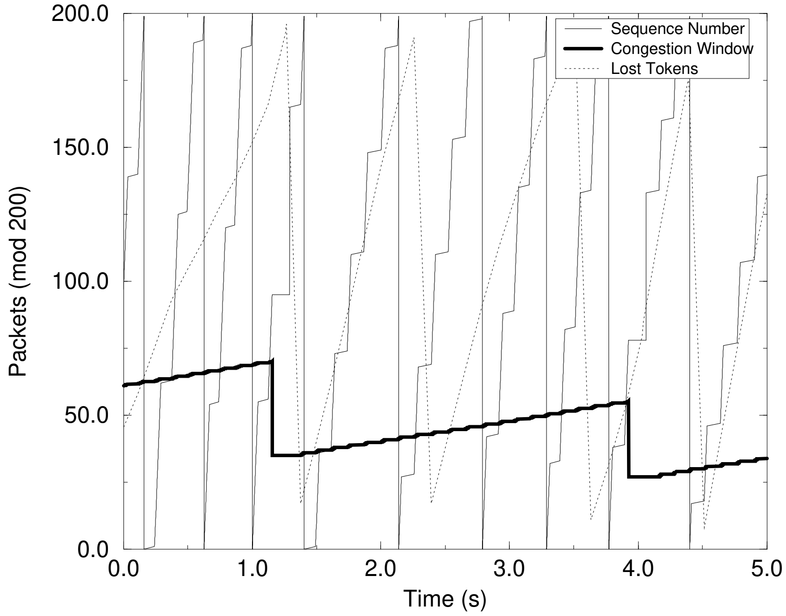
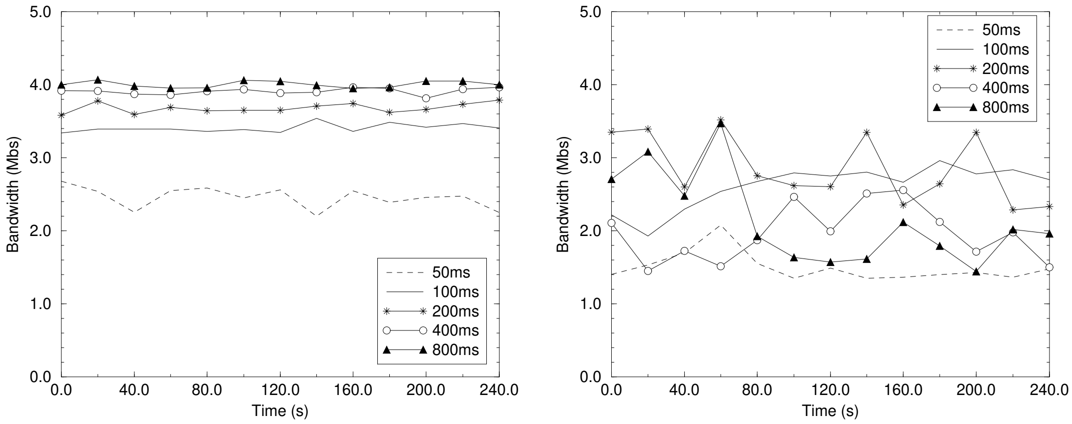


Figure 5.3: Packet trace of $4Mbs$ connection.

and enforces an advertised window (AWND) as a measure of its buffering capacity. The sender enforces a congestion window (CWND) as a measure of the capacity of the network. The sender is prohibited from sending more than the minimum of AWND and CWND worth of unacknowledged data. As described in Chapter 2, when the loss of a segment is detected, TCP cuts its congestion window in half or sets its congestion to 1 depending on whether its fast recovery or its retransmission timeout mechanism is used. For connections with reservations, both actions are overly conservative behavior since they are insensitive to the reservation that a particular connection may have. Thus, even when tokens are present and the sender is eligible to transmit a new segment, it may be throttled by the congestion window. As shown in Figure 5.3, the rate of token loss increases (as indicated by the change in slope in the lost token curve) when a packet loss is detected (as indicated by the decrease in congestion window), and slowly decreases as the congestion window opens up.

Another cause for token loss is the presence of persistent gaps in the acknowledgment stream. Such gaps are part of a phenomenon commonly referred to as ACK-compression [70]. Since TCP uses acknowledgments to trigger transmissions, any significant time gap between the receipt of successive acknowledgments causes the token bucket to overflow and results in a loss of transmission credits. The effects of these gaps can be seen in many places in the trace where the sequence



(a) One-way traffic. (b) Two-way traffic.

$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$

Figure 5.4: Compliant throughput of $4Mbs$ connection over various token buckets.

number is frozen. There are several ways in which these gaps can develop. One is through the recovery process after a loss is detected using TCP's fast recovery and fast retransmit mechanisms. After detecting a loss (by the receipt of a given number of duplicate acknowledgments), TCP cuts its congestion window in half by halting additional transmissions until one half of the original window's packets have cleared the network. Freezing the sender for this period of time causes the token bucket to overflow, but more importantly, puts a gap in the data stream which results in a gap in the acknowledgment stream during the next round-trip interval. Gaps in the acknowledgments cause the token bucket to overflow and cause gaps in the data stream once again. Another way they can form is through the normal dynamics of network traffic. Congestion on the forward and/or reverse path, as well as additional queueing delays and jitter experienced as new connections come on-line, can also create significant gaps in the stream.

5.3.2 Effect of token bucket depth

One way to alleviate the problem of token loss is to use a deeper token bucket. To investigate the impact of the token bucket depth on compliant throughput, the experiment described in the last section was repeated across a range of token bucket depths. Figure 5.4(a) shows the compliant throughput seen by the connection with a $4Mbs$ reservation for token bucket sizes of $50ms$, $100ms$, $200ms$, $400ms$, and $800ms$ using the same network topology and traffic. Increasing the token bucket depth improves the compliant throughput seen by a connection. However, it is only when the token buckets are very large ($400ms$ and $800ms$ in this case) that the compliant throughput seen by a connection remains at the reserved rate. Unfortunately, for a $4Mbs$ connection, this bucket depth corresponds to a maximum burst of compliant packets of up to $200KB$. In order for the network to ensure that compliant packets are not dropped, it must have the capacity to buffer such bursts. Without sufficient buffer space, a significant amount of burst losses can occur, causing the performance of the TCP connection to deteriorate. To see the effect of this, the traffic going through the network is increased by adding identical traffic going in the reverse direction. That is, all connections are bidirectional. Adding traffic has several effects. One effect is that it spaces out acknowledgments even further. Another, more problematic effect, is that it adds more congestion to the network which causes queues to be fully occupied. Without sufficient buffer space to handle large token buckets worth of priority packets, reserved connections experience a substantial amount of burst losses. This causes the connection to eventually freeze since recovering from multiple losses using Reno TCP, in particular, requires multiple round-trip times. Because of this, the source is eventually throttled until the lost packets are successfully retransmitted [17]. Note that as the bandwidth-delay is increased, the advertised window becomes a limiting factor in the performance of the source. If the advertised window is not large enough, whenever a single packet is lost, the source must freeze until the packet is successfully retransmitted. Figure 5.4(b) shows the result of this experiment. In contrast to Figure 5.4(a), large token buckets do not give any additional performance improvement. The connection never receives a compliant throughput more than half of its $4Mbs$ reservation.

The use of large token buckets allows large bursts of marked packets into the network which can result in loss of marked packets, thus defeating the service differentiation mechanism provided by ERED. In a WFQ implementation of controlled-load service, this is akin to overflowing a flow's queue by allowing it to burst at a rate greater than the queue length. As with any service differentiation, in order to ensure correct behavior, admission control must be done with the sources to guarantee performance. For ERED queues, this essentially means that the min_{th} and max_{th} values must be set appropriately so that marked packets are not dropped due to an over-occupation of unmarked packets. For a given queue size there is some flexibility in terms of setting the thresholds depending on the traffic load. As the load due to controlled-load traffic increases, the max_{th} and

min_{th} values can be lowered to ensure low or no loss of conformant controlled-load traffic. As the controlled-load traffic load decreases, the max_{th} and min_{th} values can be set higher to improve the throughput of best-effort traffic. However, the extent to which this flexibility in setting the queue thresholds can be used to ensure correct behavior is limited. As discussed in Section 5.2, for a given queue size and link speed, the aggregate controlled load traffic that can be admitted into the system is limited by

$$\left(\sum_{i=1}^{n} r_p^i - L\right) \times \frac{max_{th}(unmark)}{L} \leq B - max_{th}(unmark).$$

The above condition guarantees zero loss of conformant controlled-load traffic in the worst case scenario where all controlled-load sources burst traffic at the highest rate at the same time. In the rest of this chapter, it is assumed that the above condition is satisfied at every node and that marked packets are never dropped. Note that the scenario captured in the above inequality is very pessimistic. It is extremely unlikely for all controlled-load sources to burst at the same time. Also, guaranteeing zero loss of conformant traffic is not a requirement for controlled-load service. Consequently, it is possible to relax this condition and admit more controlled-load sessions than deemed possible by the above inequality. Empirical and statistical admission control mechanisms, such as measurement based admission control [39], can easily be used to operate the network at a high utilization while maintaining low loss rates for conformant controlled-load traffic.

5.4 TCP Adaptations

In this section, a number of modifications to TCP's control mechanisms are proposed and evaluated. These refinements help TCP adapt better in an integrated services environment.

5.4.1 Timed transmissions

Since deeper token buckets require larger buffers in routers and allow less flows to be admitted, it is desirable to keep the size of the token buckets small. To alleviate the effects of persistent gaps in acknowledgment without increasing the token bucket depth significantly, two different schemes are considered: *delayed* and *timed* transmissions. These schemes better adapt the acknowledgment-based transmit triggers to the rate-based marking mechanisms. In the delayed transmission scheme,

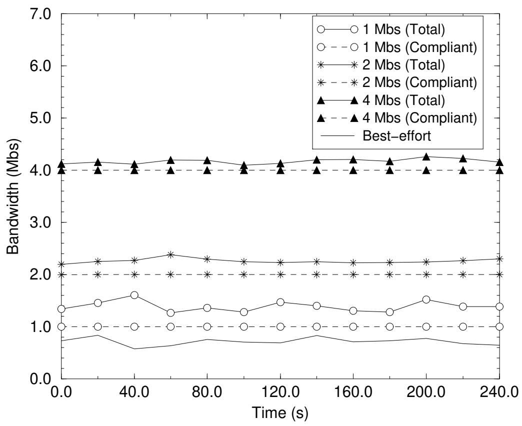
```
(1) After every acknowledgment
        if (room under congestion and advertised windows)
                if (tokens available > packet size)
                        send packet as marked
                else
                        send packet as unmarked

(2) After every TIMER expiry
        if (room under advertised window)
                if (tokens available > packet size)
                        send packet as marked
        reset TIMER
```

Figure 5.5: Timed transmission algorithm.

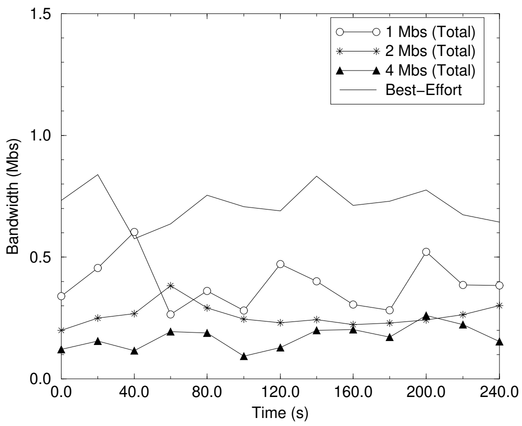
a segment is held back for a random amount of time when there are insufficient tokens present to transmit it as a marked packet. This, in effect, adds randomization to the data stream of the connection which can potentially eliminate persistent gaps in the acknowledgment stream. In addition, this scheme reduces the probability of the packets getting dropped inside the network since holding back packets increases the probability that they are sent as marked packets. While the delayed transmissions work reasonably well when the reverse path is lightly-loaded [17], additional experiments have shown that it is not very effective in the presence of reverse path congestion.

The second mechanism examined involves the use of a periodic timer. In this scheme, TCP's acknowledgment-triggered transmissions are augmented with a timer-triggered transmission mechanism. This timer-based triggering ensures that transmission opportunities are not lost while the connection is waiting for an acknowledgment. In the timed transmission mechanism, each reserved connection uses at most one timer which can have an interval which is customized. Connections can also share a single timer depending on the overhead on the end host. In the timed transmission scheme, the acknowledgment-clocked transmission algorithm is left unmodified. However, whenever a periodic timer expires, the connection examines the tokens in the token bucket. If there are sufficient tokens available in the token bucket and there is room under the advertised window of the receiver, the sender transmits the packet as marked, temporarily ignoring the value of the congestion window. The timer is then reset to wake up another timer interval later. Figure 5.5 presents the algorithm formally.

The intuition behind timed transmission is very simple. If there are enough tokens in the bucket,



(a) Total and compliant throughput.



(b) Share of excess bandwidth.

$$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$$
$$BucketDepth = 50ms, TimerInterval = 20ms$$

Figure 5.6: Throughput with timer-triggered transmissions.

as per contract with the network, the sender is eligible to inject new data in the network. Hence, the congestion window is temporarily disregarded. Note that it is also possible to disregard the congestion window for conformant sends triggered with an acknowledgment. However, the use of the timers helps prevent sending back-to-back packets, making the resulting traffic stream slightly smoother and more network-friendly. Regardless of how compliant sends are triggered, the connection still adheres to the advertised window constraint to avoid overflowing the receiver's buffers. In case of network disruption, the sending TCP freezes when the number of unacknowledged packets reaches the advertised window. Thus, the timer-triggered sends do not continue to occur in the presence of network failure. Having a timer trigger transmissions alleviates the problem of lost tokens caused by gaps in the acknowledgments. In order to guarantee zero token loss, the timer interval should be equal to $\frac{[BucketSize-(PacketSize-1)]}{BucketRate}$. This takes care of the worst case where there are $PacketSize - 1$ tokens in the bucket when a timer interrupt occurs.

Using this timer mechanism, the experiment in Section 5.3 was repeated. For the experiment, token buckets of depth $50ms$ were used along with a timer granularity of $20ms$. Figure 5.6(a) plots the total bandwidth received by all connections and the compliant bandwidth received by the connections with reservations. As shown in the figure, each connection gets its reserved rate and a share of the excess bandwidth.

While the timed transmissions allow for temporary violations of the congestion window to occur, non-compliant packets are sent only when there is room under the congestion window. Thus, this mechanism does not alter the way TCP's congestion window is calculated. Using TCP's windowing algorithm can be a problem since upon detection of a loss, the congestion window is cut in half or reduced to 1 regardless of a connection's reservation. Thus, although the timed transmission mechanism allows the connection to receive its reserved rate, TCP's windowing mechanism can restrict the controlled-load connections from competing for the excess bandwidth in the network[3]. Figure 5.6(b) plots the throughput seen by a best-effort connection and the non-compliant throughput seen by each of the reserved connections using timed transmissions. The plots show that connections with reservations receive a smaller share of the residual capacity when compared to the best-effort connection. The connections with larger reservations are penalized to a greater extent since halving the congestion window of a connection with $4Mbs$ reservation has a more drastic impact than halving the congestion window of a $1Mbs$ connection.

5.4.2 Rate adaptive windowing

For reserved connections, the swings in the congestion window should always be above the window guaranteed by the reserved rate. To account for and exploit the reservation, TCP's windowing algorithm is modified. The key idea behind this modification is that for reserved connections, CWND consists of two parts: a reserved part equal to the product of the reserved rate and the estimated round-trip time, and a variable part that tries to estimate the residual capacity and share it with other active connections. Note that the reserved part of CWND is a function of the round-trip time. While the algorithm currently uses the common TCP round-trip measurements to estimate this, measurements using the TCP timestamps option (RTTM) [36] can provide a more accurate estimate.

Assuming that the size of the reservation window is RWND, the size of the variable window is CWND $-$ RWND. In the modified scheme, the size of the variable window is adjusted using the traditional TCP windowing mechanism and simply added to the calculated value of RWND. Specifically, the sender, instead of reducing CWND by half at the beginning of the fast recovery, sets it to RWND + $\frac{\text{CWND}-\text{RWND}}{2}$. At the beginning of a slow start after detection of a lost segment through the

[3]Non-conformant controlled-load traffic is treated as best-effort traffic. Hence, residual network capacity should be fairly shared between best-effort traffic and non-conformant controlled-load traffic.

(1) After every new acknowledgment

if ($\text{CWND} < \text{SSTHRESH}$)

$$\text{CWND} = \text{CWND} + \frac{\text{CWND}-\text{RWND}}{\text{CWND}}$$

else

$$\text{CWND} = \text{CWND} + \frac{1}{\text{CWND}}$$

(2) When NDUP exceeds a threshold

$$\text{CWND} = \text{RWND} + \frac{\text{CWND}-\text{RWND}}{2} + \text{NDUP}$$

$$\text{SSTHRESH} = \text{RWND} + \frac{\text{CWND}-\text{RWND}}{2}$$

(3) Upon RTO

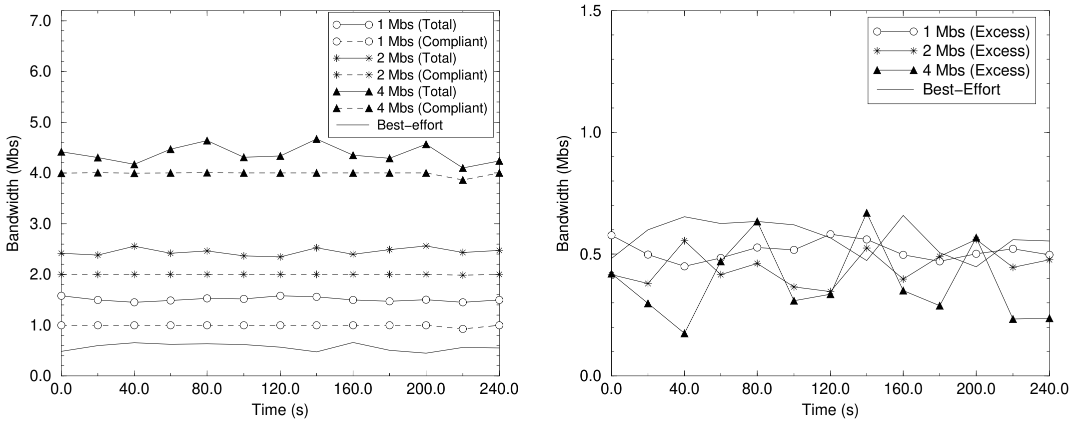
$$\text{CWND} = \text{RWND} + 1$$

$$\text{SSTHRESH} = \text{RWND} + \frac{\text{CWND}-\text{RWND}}{2}$$

Figure 5.7: Rate adaptive windowing algorithm.

retransmission timeout, it sets CWND to RWND+1 instead of 1. In both cases, SSTHRESH is set to the minimum of $\text{RWND}+\frac{\text{CWND}-\text{RWND}}{2}$ and AWND instead of the minimum of $\frac{\text{CWND}}{2}$ and AWND. Finally, because packets sent under RWND should not clock congestion window increases, window increases are scaled by $\frac{\text{CWND}-\text{RWND}}{\text{CWND}}$. Note that even with these modifications to the windowing algorithm, the sender must still adhere to the AWND restriction. That is, it is prohibited from sending more than the minimum of AWND and CWND worth of unacknowledged data. Because of this, the size of the receiver's buffer must be at least the size of the reservation window in order to sustain the reserved rate using TCP. This control algorithm is summarized in Figure 5.7.

The experiments described in the previous section were repeated with the windowing modifications in place. Figure 5.8(a) shows the aggregate and compliant throughput seen by each reserved connection using the modifications to the windowing algorithm. It also shows throughput seen by a best-effort connection between the same source and destination. As seen in the figure, all connections perform as expected. Figure 5.8(b) plots the amount of excess bandwidth received by each reserved connection, as well as the bandwidth received by the best-effort connection. When



(a) Total and compliant throughput. (b) Share of excess bandwidth.

$$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$$
$$BucketDepth = 50ms, TimerInterval = 20ms$$

Figure 5.8: Throughput with timer and windowing modifications.

compared to Figure 5.6(b), the reserved connections obtain a fairer share of the excess bandwidth.

A common concern with any modification to TCP's windowing mechanism is that the change may be too aggressive and thus, cause unnecessary congestion. The experiments which have conducted so far, including the ones reported in this chapter, show no bias towards connections using the modified windowing mechanism. A number of different flavors of the windowing algorithm have also been examined. They differ in the way RWND is computed and CWND is clocked. RWND is computed by multiplying the reserved rate with the estimated round-trip time. Depending on how conservative the windowing mechanism needs to be, different estimates of round-trip time can be used. Experiments using both the best and average estimates of round-trip times were also performed and showed similar results. Note that in times of congestion, the estimated round-trip time tends to be large and thus, the rate-based window can also grow large during a period of time when the network needs a respite. Using the best observed round-trip time in this case, allows the connection to be on the conservative side in calculating its rate-based window.

Another concern in deploying the modifications is that it may potentially lead to congestion collapse since sources maintain a minimum sending rate which they do not back off from. In order for such modifications to be deployed, the use of proper signaling, admission control, and resource reservation must be required in order to prevent congestion collapse. Even when such mechanisms

CPU Type	$133MHz$ PowerPC	$33MHz$ POWER
Timer setting	7.4μs	14.0μs
Timer handling	7.1μs	30.1μs
Timer canceling	6.5μs	9.6μs

Table 5.1: Timer overheads (AIX 4.2 kernel).

exist, it still may be necessary to add some mechanism to back off the timer and windowing modifications. For example, misconfiguration and/or the presence of legacy equipment may make it impossible to guarantee an end-to-end minimum sending rate. In order to prevent congestion collapse in these scenarios, the end host should be modified to respond to a large amount of packet loss. One simple alternative is for the source to simply turn off both the timer and windowing modifications whenever it detects any loss of marked (compliant) packets.

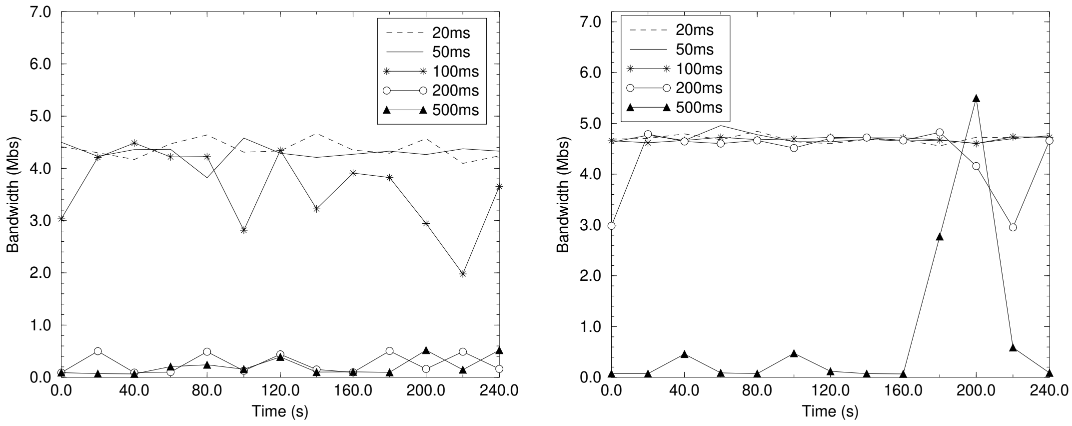
5.5 Fine-Grained Timers

This section explores the cost associated with deploying fine-grained timers into TCP as well as the benefits of using such a timer for sending data.

5.5.1 Timer overheads

In the description of the timed transmission algorithm, the existence of connection-specific timers is assumed. However, it is possible, and desirable, to use a common timer shared amongst all reserved connections. Such optimizations can be easily incorporated using techniques such as the protocol timer services in the BSD-style TCP/IP stack. One of the common criticisms against the use of timers is the overhead associated with handling timer interrupts. For that reason, TCP uses coarse-grained (typically $200ms$ and $500ms$) timers. However, the state of the art in processor technology and operating systems has advanced considerably since the first design and implementation of TCP. Processors and timer implementations today are much faster, and consequently, the overheads of handling timer interrupts are much lower.

Table 5.1 shows the overheads of setting, canceling, and handling timers in two IBM RS/6000



(a) $80KB$ buffers. (b) $160KB$ buffers.

$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$

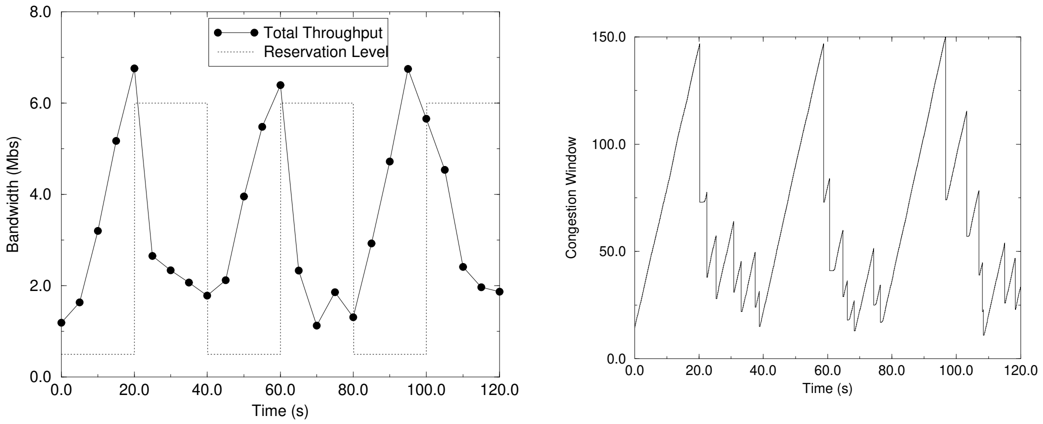
$BucketDepth = TimerInterval + 30ms$

Figure 5.9: Throughput of $4Mbs$ connection over various timer intervals.

machines running AIX 4.2, one equipped with a $33MHz$ POWER CPU and the other with a $133MHz$ PowerPC CPU. The table shows that the overheads of timer operations in modern systems ($133MHz$ PowerPC) are quite small. Even when older systems, such as the $33MHz$ RS/6000, are considered in this study, the overheads are well within acceptable limits. Note that these measurements were taken without any optimization to the timer data structures in the AIX kernel. In AIX 4.2 timer blocks are arranged in a linear array. The overhead of timer operations are expected to be even lower if the timer blocks are stored as a hash table. However, at this point such an optimization is not deemed necessary.

5.5.2 Buffer requirements

While there are concrete costs associated with using fine-grained timers, there are also significant benefits. One benefit in using these timers is that it reduces the size of the token buckets used for each application. From the calculations in Section 5.4, given a certain timer interval, the token bucket depth should be at least $(TimerInterval \times BucketRate) + (PacketSize - 1)$ to prevent token loss. Because the token bucket size grows linearly with the timer interval, using fine-grained timers allows applications to request smaller token buckets. Since each router must be able to buffer a burst the size of the entire token bucket for each application, the size of these buckets has a direct



(a) Throughput (b) Congestion window trace

$BucketDepth = 800ms$

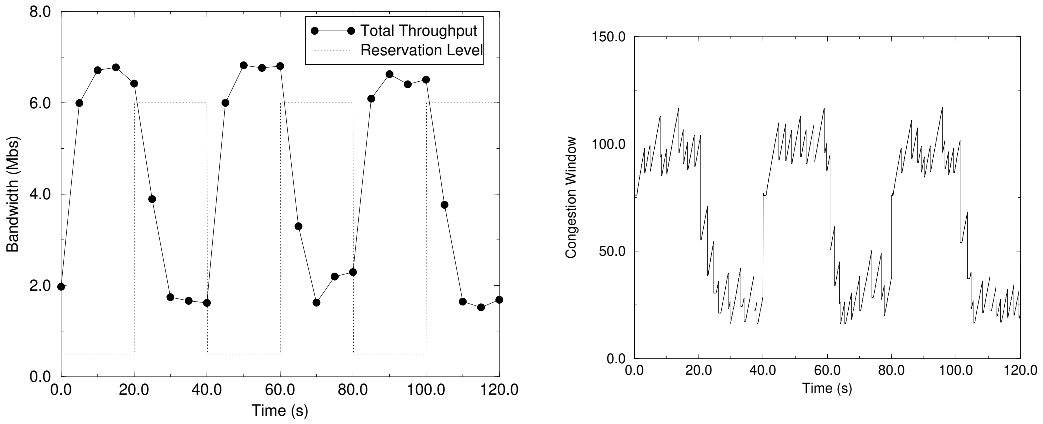
Figure 5.10: Dynamics of unmodified TCP.

impact on the amount of buffer space required in network routers.

Figure 5.9 shows the impact that the timer-interval has on the throughput of the $4Mbs$ connection using the same multi-hop network setup. The simulations were run using both the timer and windowing modifications as described in Section 5.4. As Figure 5.9(a) shows, as the timer interrupt interval increases, the throughput of this connection drops considerably. The reason why this drop is so dramatic is that the lack of buffer space in the network causes a significant amount of burst losses. Burst losses severely limit the throughput of Reno TCP-variants since it takes one round-trip time to recover from each loss. This causes the sending TCP to degenerate into a stop-and-wait protocol. Figure 5.9(b) shows the results of the same experiment using buffers which are twice the size ($160KB$). With significantly larger buffers, the connection is able to get its share of the bandwidth over a larger range of timer interrupt intervals.

5.6 Transient Behavior

To take a closer look at the transient behavior, a controlled-load connection's reservation is changed in the presence of several best-effort connections. In particular, a controlled-load connection which has a reservation that is toggled from $0.5Mbs$ to $6Mbs$ every 20 seconds was run between $n0$ and $n5$. Four pairs of best-effort connections were also run between these two nodes.



(a) Throughput (b) Congestion window trace

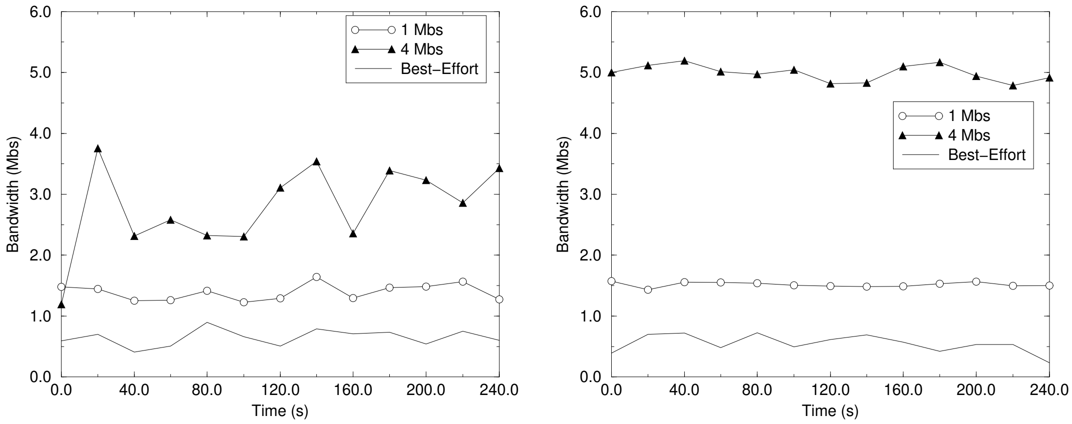
$BucketDepth = 50ms, TimerInterval = 20ms$

Figure 5.11: TCP with timer and windowing modifications.

Figure 5.10(a) plots the total throughput of the controlled-load connection using unmodified TCP. This connection uses a token bucket depth of $800ms$ in order to prevent a large amount of token loss[4]. The plot also shows the reservation that the connection has over time. As shown in the figure, the bandwidth received by the connection reacts slowly to increases in the reservation while it reacts quickly to the decrease in reservation. This is directly attributed to the additive increase/multiplicative decrease property of TCP's windowing algorithm [34]. Figure 5.10(b) shows the congestion window trace for the normal TCP source. The graph shows the congestion window linearly increasing in response to an increase in reservation level at times $t = 0s$, $t = 40s$, and $t = 80s$. Thus, by the time the window size reaches a size which can support the size of the reservation, almost the entire 20 second interval has elapsed. This is why the total throughput of this connection lags behind the reservation change.

Figure 5.11(a) shows the same experiment, but with the reserved connection using the timer and windowing modifications described earlier. Note that the throughput of the connection immediately reacts to both the increase and decrease in reservation levels. Figure 5.11(b) shows the congestion window trace of the connection over the same time period. The window size in this case reacts more quickly to the change and thus allows the connection to get its reserved rate. One advantage of rate-based windowing is that the congestion window immediately reflects any changes in reservation

[4]Note that the size of the buffers on each interface is $100KB$, which is enough to absorb the large bursts that are caused by the deep token bucket.



(a) Reserved connections using Reno (b) Reserved connections using SACK

$Q_{size} = 100KB$

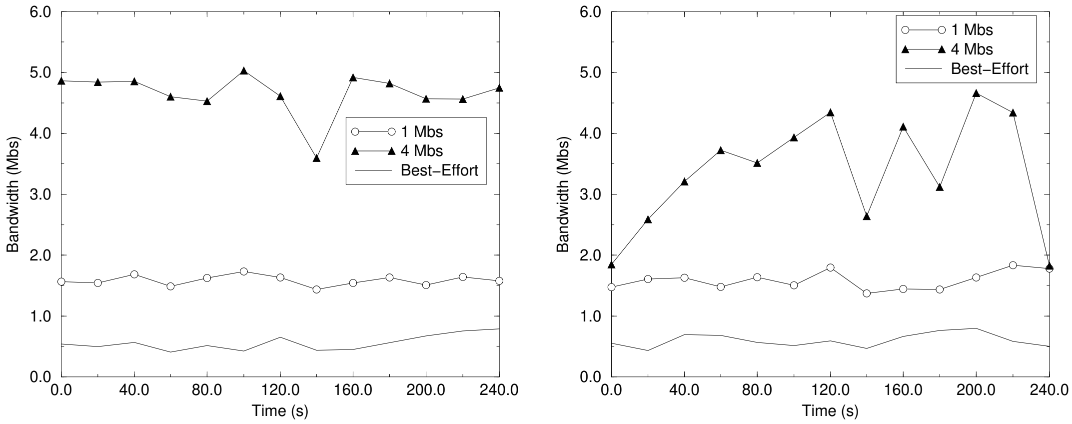
Figure 5.12: All nodes using drop-tail queues.

level, especially when the level is increased. For TCP-variants which use fast retransmit and fast-recovery, the connection is often in a state where it is increasing its congestion window additively in order to find a congestion window that indicates the amount of available bandwidth in the network. With a large change in reservation, this can take a fairly long time, especially for connections with large round-trip times.

5.7 Path of Evolution

For the experiments reported in previous sections, it is assumed that all routers on the path of a connection employ support for AF such as ERED. While this is desirable, modifications to the Internet infrastructure will be incremental and evolutionary. To understand the impact of this heterogeneity in the network, a number of scenarios are considered where none or only a selective subset of routers are ERED-capable.

The purpose of the first experiment is to examine the impact of non-ERED gateways on reserved connections. In this experiment, two TCP connections with reservations of $1Mbs$ and $4Mbs$ as well as a single best-effort connection are run between nodes $n0$ and $n5$. Pairs of best-effort connections are also run between nodes $n6$ and $n7$ and between $n8$ and $n9$. Figure 5.12(a) plots the throughput seen by these connections when all the routers are drop-tail routers which do not distinguish between marked and unmarked packets. The connections with reservations use the timed transmission



(a) Bottleneck links using ERED (b) Non-bottleneck links using ERED

$min_{th} = 20KB, max_{th} = 80KB, Q_{size} = 100KB$

Figure 5.13: Effect of selective reservation.

mechanism and the modified windowing scheme. Note that while no service differentiation is being done on the links, it is assumed that adequate admission control and provisioning is being done in order to ensure that the offered load does not overwhelm network links and cause congestion collapse.

As shown in Figure 5.12(a), the reserved connections indeed see higher throughput than the best-effort connection between the same nodes. However, the absence of ERED mechanisms in routers adversely impacts the performance of connections with reservations. This situation worsens with the increase in the reservation level. This is because a connection with a higher reservation transmits a larger amount of data and thus, the number of packet drops it experiences is proportionally higher. In addition, the connection with a larger reservation transmits data in larger bursts and is susceptible to burst losses in the drop-tail queues. This is reflected in the oscillation in the bandwidth curve for the connection with a $4Mbs$ reservation. Burst losses, as described earlier, interact poorly with Reno sources since it takes a full round-trip time to recover from each loss. This freezes the sender and prevents it from getting its reserved rate. A packet trace of this connection verifies this behavior. Figure 5.12(b) shows the same experiment with the reserved connections using SACK TCP. As shown in the figure, despite the absence of ERED queueing, all connections manage to achieve aggregate throughputs close to their respective desired levels.

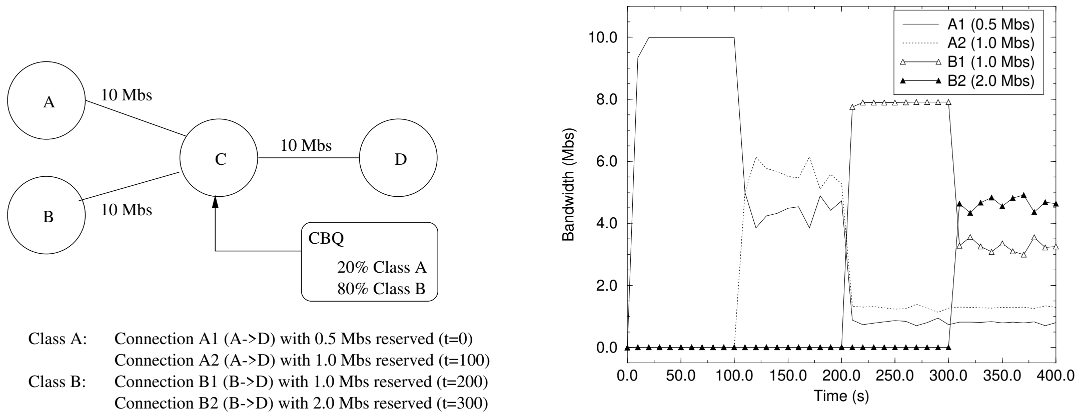
From the results of these experiments, it is fair to conclude that even without sophisticated scheduling mechanisms it may be possible to extend the paradigm of equal sharing of network capacity to one where it is shared in accordance with allocations. Note, however, that in the experiments here, the aggregate reservation is lower than the network capacity. Cooperation among different TCP connections is also assumed. This means that admission control and voluntary adherence to the socially cooperative congestion control is still required. The ERED mechanism simply provides additional protection to compliant controlled-load traffic against best-effort and non-compliant controlled-load traffic.

To examine the impact of ERED-capable routers at selected places, the experiments described above were repeated using Reno sources with only a few selected routers employing ERED. Figure 5.13(a) shows the throughput of the same set of connections when interfaces only at the bottleneck links between $n1$ and $n2$ and between $n3$ and $n4$ employ ERED queues while the rest of the interfaces use drop-tail queues. As the graph shows, placing ERED queues at bottleneck points in the network is sufficient to provide connections their allocated share of the bandwidth. However, as shown in the figure, the connection sometimes dips below its reservation level due to burst losses which can occur in the presence of drop-tail queues. Figure 5.13(b) shows the throughput of the connections when the ERED queues are placed on non-bottleneck links between $n0$ and $n1$ and between $n4$ and $n5$. As with the drop-tail experiments in Figure 5.12(a), the performance of the high bandwidth connection suffers throughout.

The results from these experiments demonstrate that there is an effective path of evolution of integrated services in the Internet. For the integrated services to be useful, it is not required to upgrade the entire infrastructure at the same time. There is substantial value in following an evolutionary path where at first the control mechanisms at the end hosts are modified and routers support admission control. Enhanced queueing mechanisms, such as ERED, can then be deployed at observed bottlenecks and then gradually throughout the network.

5.8 CBQ and ERED

The experiments in this chapter have shown how to effectively provide minimum bandwidth guarantees in the network using ERED gateways. While this service may be useful to a large class of applications, in a fully-evolved integrated services Internet, such a mechanism must coexist with



(a) Network topology. (b) Throughput.

Figure 5.14: CBQ experiment.

other mechanisms for providing a range of alternative services. This allows applications written using such a service, to continue to work as the Internet infrastructure is upgraded and more sophisticated packet and link scheduling support is put into place.

The ERED mechanism can be easily embedded into more sophisticated scheduling disciplines such as class-based queueing (CBQ) [27]. CBQ is one of the more popular mechanisms proposed for packet and link scheduling in an integrated services Internet. In CBQ, datagrams belonging to different classes are put in different queues in the routers. The queues are serviced in different priority order based on the allocation given to the associated traffic class. Embedded as a class in CBQ, ERED can be used to provide weighted bandwidth sharing between connections of a class. By aggregating connections with varying bandwidth requirements in one class, the total number of classes in a class-based queue is reduced and thus, the overhead in link scheduling. To examine this possibility, the ERED queue was embedded into the CBQ implementation of `ns`. Its performance in the network shown in Figure 5.14(a) was then examined. This network consists of two agencies, A and B, who share a common link (between nodes C and D) to a provider's network. In this setup, agency A is entitled to 20% of the link's bandwidth while agency B is entitled to the remaining 80% of it. Node C uses CBQ with 20% of the link share allocated to traffic from agency A, and 80% allocated to traffic from agency B. Note that while either one of the two agencies is idle, the other, active agency, is entitled to use the entire link for itself. In addition, since the allocation is relative, admission control must assume the worst case scenario in admitting flows to a class. That is, it must assume all other classes are using their share of the link. Both queues within the CBQ system use the ERED mechanism to share bandwidth between individual connections.

Figure 5.14(b) shows the throughput seen by connections originating from A and B and traversing the link between C and D. Connections $A1$ and $A2$ originate from agency A and have reserved rates of $0.5Mbs$ and $1Mbs$, respectively. They start at times $t = 0s$ and $t = 100s$. Connections $B1$ and $B2$ originate from agency B and have reserved rates of $1Mbs$ and $2Mbs$. These connections start at times $t = 200s$ and $t = 300s$, respectively. As the graph shows, between $t = 0s$ and $t = 100s$, connection $A1$ gets all of the bandwidth since it is the only active connection. Between $t = 100s$ and $t = 200s$ (after connection $A2$ starts) the link's bandwidth is shared between connections $A1$ and $A2$. However, since $A2$ has a $1Mbs$ reservation, it gets slightly more total bandwidth than $A1$. When $B1$ starts at $t = 200s$, it is the only active connection from agency B. Hence, it receives the entire 80% of the link's bandwidth ($8Mbs$). The two connections from agency A then share the remaining bandwidth ($2Mbs$) according to their reservations. Finally, at $t = 300s$, connection $B2$ starts and the $8Mbs$ allocated to agency B is split between connection $B1$ and $B2$ in accordance with their reservations. That is, $B2$ gets approximately $1Mbs$ more than $B1$. What happens throughout the course of this experiment is that when the class is allowed to be overlimited, the ERED queue is drained at a sufficient rate so as to support higher rates of input data. As soon as the class becomes regulated, the queue builds up, the ERED queue drops unmarked packets and the connections in the class resumes sending at a lower rate.

5.9 Conclusions

This chapter has examined a way of providing a large class of bandwidth-sensitive Internet applications with a useful service using minimal enhancements to the network infrastructure. Towards this end, a simple extension to the packet queueing algorithms at the routers has been proposed and analyzed. Assuming a network which supports minimum rate guarantees, a number of modifications to TCP's congestion control algorithm have been proposed and evaluated. These modifications take advantage of the network support provided and can allow connections with reservations to obtain their reserved rates and share excess bandwidth in the network. It is important to note that these modifications require the presence of end-to-end signaling, admission control, and resource reservation. Without such support, it is necessary for sources to turn off the modifications in order to prevent possible congestion collapse in networks which do not support minimum rate guarantees.

The study reported in this chapter can be extended in many ways. In particular, applying this work in the context of other transport protocols is being considered especially with RTP and UDP. Many multimedia applications do not require the reliable delivery that TCP provides. While this study focuses on TCP, implementing a similar scheme using RTP and UDP fitted with TCP's flow-control mechanism is possible. Another extension being considered is the use of BLUE-based mechanisms in place of ERED to do service differentiation. As shown in Chapter 4, BLUE can manage congestion with a minimal amount of buffer space. Enhancing BLUE with additional priority-based mechanisms is relatively easy and can provide significant improvement in performance. Another key area of future work is the admission control policies for such a service. While this chapter has not addressed admission control, the observations made on token bucket depths, router buffer sizes, source burstiness, and ERED-parameterization will be instrumental in developing admission control policies for this service.

On a final note, any allocation-based sharing of network capacity has to be associated with a policy and/or pricing scheme. The proposal for prioritizing a part of a traffic stream with marking and competing with best-effort traffic for sharing the residual capacity fits in very well with pricing schemes which are currently being considered for the Internet. Users paying for a certain level of marking see incrementally better performance over those who do not. During times of light loads, when the incremental costs of congestion are low, the user can decrease his/her level of bandwidth reservation and costs until an acceptable level of aggregate throughput is observed.

CHAPTER 6

ADAPTIVE PACKET MARKING FOR PROVIDING DIFFERENTIATED SERVICES IN THE INTERNET

6.1 Introduction

The DIFFSERV architecture does away with the problem of maintaining and managing flow states in the core of the network. However, in order to provide firm service assurances, one still needs to provision the network to handle the offered load. As described in Chapter 5, one way to keep the offered load from exceeding the provisioned capacity is to assign traffic profiles to users and networks and then monitor and enforce them [11, 19, 31, 37] at the user-network and network-network interfaces. Such approaches that provide firm guarantees on performance require end-to-end signaling in order to communicate the traffic profiles throughout the network. They also require policing and shaping to enforce the traffic profiles at the network boundaries. This chapter describes an alternative approach to service differentiation that provides soft bandwidth guarantees, but eliminates the need for end-to-end signaling and enforcement of traffic profiles.

This chapter considers a network service model which uses DIFFSERV mechanisms to deliver soft bandwidth guarantees to applications using a modest enhancement to the congestion control and queue management algorithms provided by today's Internet. As in Chapter 5, the network is assumed to support a one-bit priority scheme with lower loss rates for higher priority traffic as in AF. In this model, traffic is monitored at both user-network and network-network interfaces. However, instead of strictly allocating and enforcing traffic profiles on an end-to-end basis, a more flexible model that relies on adaptive traffic control at the host and at the edges of the network is used. In this model, the user or network administrator specifies a desired minimum service rate for a connection or connection group and communicates this to a control engine located at or near the host-network interface. The objective of the control engine, which is called a *packet marking engine* (PME), is to monitor and sustain the requested level of service by setting the DS field in the packet headers appropriately. By default, all packets are generated as low priority packets. If the observed service rate at the low priority level either meets or exceeds the requested service rate, the PME assumes the role of a passive monitor. If however, the observed throughput falls below the minimum target rate, the PME starts prioritizing packets until the desired target rate is reached. Once the target is reached, it strives to reduce the number of priority packets without falling below the minimum requested rate. In this architecture, traffic needs to be monitored and marked only at the host-network interface. However, the end host and network edge mechanisms described in this chapter are intelligent enough to adapt appropriately in network environments where packets are re-marked and/or dropped at the network-network interfaces for the purpose of enforcing bi-lateral service level agreements between providers.

As with any type of differentiated service mechanism, it is assumed that the network provides incentives that would prevent users from continually requesting the highest level of service. Usage-based pricing is an example of one such incentive mechanism. Many ISPs (Internet Service Providers), such as UUNet, PSINet, and MCI, already provide services wherein users are charged based on link utilization measured over fixed time intervals. It is rather simple to extend this pricing model to levy higher prices for the high priority traffic. Such a pricing mechanism would encourage judicious use of priority service based on application requirements and usage policies. While pricing is not the focus of this study, one of the key advantages of the proposed architecture is that it can provide simple mechanisms for calculating near-optimal prices based on congestion costs [43].

In the following sections, the efficacy and the robustness of the proposed framework is demonstrated. Using extensive simulations, the proposed architecture is shown to adapt with the traffic dynamics in the Internet and can thus eliminate the risk of congestion collapse. When used in conjunction with intelligent queue management, it can also identify and penalize non-adaptive and/or malicious flows and hence provides sufficient incentives for applications to be well-behaved. Finally, a number of deployment issues are discussed.

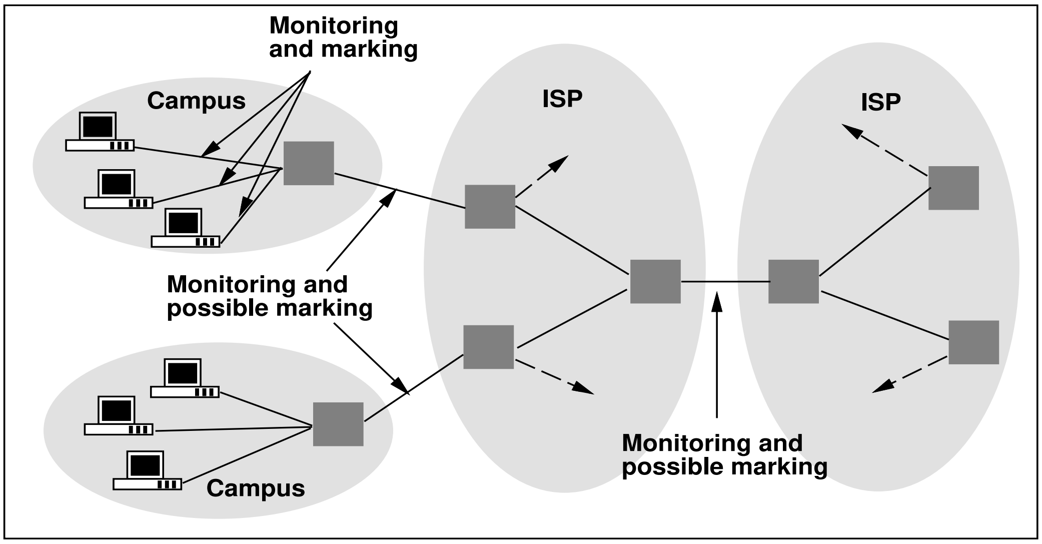


Figure 6.1: Packet marking scenarios

6.2 Service Model

This section presents a brief overview of the service architecture used. As mentioned earlier, the objective of this work, in contrast to the work in Chapter 5, is to develop a differentiated services framework without using end-to-end signaling and without enforcing explicit profiles on individual traffic flows at the network boundaries. Towards this end, it is assumed that the network is able to support two traffic types: priority and best-effort as embodied by the AF proposal in the IETF. In this proposal, priority information is carried in the DS field (DSCP) of the IP header and that by default, all packets are initially sent with their DS field cleared (best-effort). For reasons of simplicity, these packets are referred to as unmarked packets. Consequently, priority traffic is referred to as marked traffic. While there is no guaranteed service level associated with an AF setting, it is assumed that the higher priority generally translates into a better quality of service. In line with the Internet design philosophy and the DIFFSERV architecture, most of the intelligence in this architecture is at the edges of the network. The routers and gateways provide only modest functionality to support service discrimination, namely support for the AF PHB. Figure 6.1 shows a picture of this architecture.

Given this service model, the goal is to develop packet marking schemes which can be deployed at the host-network interface that will allow an individual connection or a connection group to achieve a target throughput specified by the user or network administrator. For example, a user may request a specific target rate for a particular connection or an aggregate rate for a group of connections. The objective of the packet marking scheme is to monitor the throughput received by the connection or connection group and appropriately adjust the packet marking so that the sustained rate is maintained satisfying all the policy constraints. Due to the particular nature of the service model, at times it may not be possible to sustain the requested target rate due to over-commitment of resources. Such lapses may also be caused by partial deployment of DIFFSERV mechanisms or oversubscription. A significant part of the effort goes into detecting such cases and taking appropriate actions whenever required.

In this service architecture, traffic flows are monitored and packets are marked at the host-network interface. However, the service architecture allows for packets to be re-marked at multiple points along the path in order to enforce different policies and service contracts. Consider for example, a campus or enterprise environment where applications running at different hosts may mark packets at certain rates to achieve their respective target throughputs. Packets may be re-marked at the boundary between the internal network and external network to enforce the service agreement with the network service provider. Similar re-marking may also occur in order to enforce a bi-lateral agreement between service providers when traffic crosses provider boundaries. While this scheme can adapt in an environment where packets are marked at multiple points, this chapter considers scenarios where packets are marked only once. The impact of packet re-marking is under investigation and will be addressed in future work.

Marking mechanisms of two different flavors are considered: (1) where the marking engine is transparent and potentially external to the host, and (2) where the marking engine is integrated with the host. In either case, the packet marking engine (PME) maintains local state that includes the target throughput requested for a connection or a group of connections. It passively monitors the throughput of a connection or the aggregate throughput of a group of connections and adjusts packet marking in order to achieve the target throughput requested by the user. Placing the PME external to the host has significant deployment benefits since it can be deployed transparently to the hosts. On the other hand, integrating the PME with the host protocol engine can provide a solution that adapts better with the flow and congestion control mechanisms used at the transport layer. In particular, the integration of the PME and the TCP control mechanisms is considered. The rest of this chapter focuses on TCP as the transport protocol of choice. However, the proposed schemes can be easily generalized to any transport protocol that is responsive to congestion in the network.

6.3 Source Transparent Marking

A PME snoops on connections passing through it and measures their observed throughputs. If the measured throughput is sufficiently close to the requested target rate, it takes the role of a passive monitor. However, if the observed throughput of a connection is lower than its requested target, the PME takes a more active role and starts marking packets belonging to the connection or connection group. The fraction of marked packets varies from 0 to 1 depending upon the measured and target throughputs. Selective marking essentially upgrades a fraction of the packets belonging to the connection to the higher priority level. The PME continually adjusts the fraction of packets to be marked in order to sustain a bandwidth close to the requested target rate, while keeping the number of marked packets as low as possible.

One of the important tasks performed by a PME is measuring the throughput seen by connections passing through it. This is fed into the packet marking process that has to adapt to the changes in observed throughput caused by variations in network load. While the overall measure of network performance from an application's point of view is goodput, the PME used in the experiments in this chapter only measures the local bandwidth consumed by a connection. It counts bandwidth against a connection or connection group when it receives a packet from it, even though the packet may be dropped later on in the transit path. One of the reasons for measuring local throughput, instead of end-to-end goodput, is simplicity. The PME does not have to understand the transport layer protocol semantics in order to determine whether or not the application's data was actually delivered. In some cases, even if the PME is well aware of the transport layer semantics, it may not have access to the stream of acknowledgments from the receiver to compute goodput. This may be the case when the forward and the return paths of connections are different. The most important reason for counting local throughput is to give incentive for end hosts to send packets which have a good chance of being delivered. Thus, a malicious or non-responsive source has its packets counted against itself regardless of whether they have been delivered.

The local throughput seen by a connection can be measured in several ways. One simple technique is to measure the amount of data transferred with a sliding window and to use the average

Every update interval:
    scale = $| \, 1 - \frac{B_o}{B_t} \, |$
    if $(B_o < B_t)$
        P_{mark} = P_{mark} + scale $\times$ $increment$
    else
        P_{mark} = P_{mark} $-$ scale $\times$ $increment$

Figure 6.2: TCP independent algorithm

bandwidth received over this window as a measure of the observed bandwidth. If the window is small, the measured throughput is biased towards the more recent observations. If window is large, the computed throughput converges to the long-term average bandwidth seen by the connection. While this is a fairly accurate and tunable measure of the observed throughput, it requires a window's worth of information stored for each connection. For the experiments reported in this study, a lightweight alternative mechanism is used. The throughput seen by a connection over a small time window is first measured and the observed bandwidth is then computed as a weighted average of this measured throughput and the current value of observed bandwidth.

6.3.1 TCP-independent marking

The most important task of a PME is to adaptively adjust the packet marking rate based on the measured throughput. In this chapter, a probabilistic marking scheme where the packets are marked randomly as they pass through the PME is considered. The marking probability (P_{mark}) is periodically updated depending on the observed bandwidth (B_o) and the corresponding target bandwidth (B_t). Figure 6.2 shows a simple algorithm designed for this purpose. As seen from the algorithm, when the observed bandwidth is less than the target bandwidth, P_{mark} is incremented in steps. Similarly, P_{mark} is decremented in steps if the observed throughput exceeds the target rate. Note that both increments and decrements in P_{mark} are scaled by the difference between observed and target throughputs. That is, the changes in P_{mark} get smaller as the observed bandwidth nears the target bandwidth. This scaling damps the amplitude of oscillations of the marking probability.

In order to understand the effect of packet marking, a simple scenario is evaluated using `ns` [46]. As shown in Figure 6.3, the simulated network consists of six nodes, $n0$ through $n5$, and five links connecting them. Each link is labeled with its respective link bandwidth and has a transmission delay of $10ms$. In order to support AF-style priority marking, ERED queues are used with min_{th} of

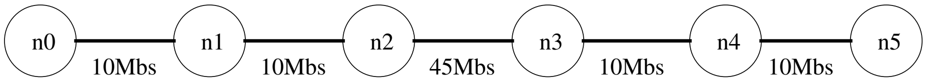
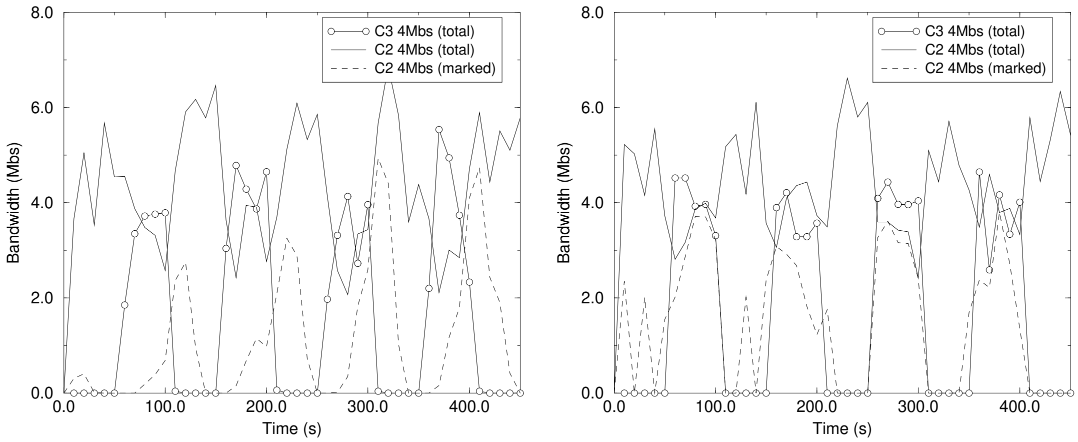


Figure 6.3: Network topology.

$10KB$, max_{th} of $80KB$, and an initial drop probability of 0.05 for unmarked packets. Three connections are then simulated between nodes $n0$ and $n5$: an infinite best-effort TCP connection (*C1*), a second infinite TCP connection (*C2*) with a $4Mbs$ target bandwidth, and a third TCP connection (*C3*) that toggles on and off every 50 seconds and has a throughput requirement of $4Mbs$ when it is on. The observed throughputs and marking probabilities are updated every $100ms$.

In this network configuration, when only *C1* and *C2* are active, the bottleneck link bandwidth of $10Mbs$ is shared evenly between them and thus, no packet marking is required for *C2* to achieve its target of $4Mbs$. However, when *C3* is active, an even share of the bottleneck bandwidth ($3.33Mbs$) does not satisfy the target throughput requested by *C2* and *C3*. The PME has to mark packets belonging to *C2* and *C3* in order for them to obtain the higher throughput. Figure 6.4(a) shows the throughputs received by *C2* and *C3* over time. In this experiment, P_{mark} is adjusted in steps of 0.01. As the figure shows, *C2* is slow in reacting to changes in the network. When all of the sources are on, it is consistently below its $4Mbs$ target bandwidth. It takes a significant amount of time to build up P_{mark} in response to the changes in the network load. Figure 6.4(a) also shows the marking rate for connection *C2*. As expected, the marking rate lags behind the changes in the network load, slowly rising in response to an increased traffic load and slowly falling in response to a decreased traffic load. To examine the other end of the spectrum, the experiment was repeated while allowing P_{mark} to be updated in steps of 1.0. That is, when more bandwidth is needed, all packets are marked. Otherwise, packet marking is turned off. Figure 6.4(b) shows the results from this experiment. As expected, in this experiment, P_{mark} adapts very quickly to the changes in the network load, thus allowing *C2* to achieve its target rate during periods of increased traffic load. This rapid response also allows the PME to turn off packet marking quickly when it detects that the available bandwidth is sufficient to satisfy the target rate. While adapting quickly to changes in network conditions has its benefits, it can also cause significant burstiness in both marked and unmarked packet streams. For example, if packet marking is turned on for a connection with a relatively high target throughput, it may cause large spikes in the number of marked packets in the



(a) P_{mark} increment = 0.01

(b) P_{mark} increment = 1.00

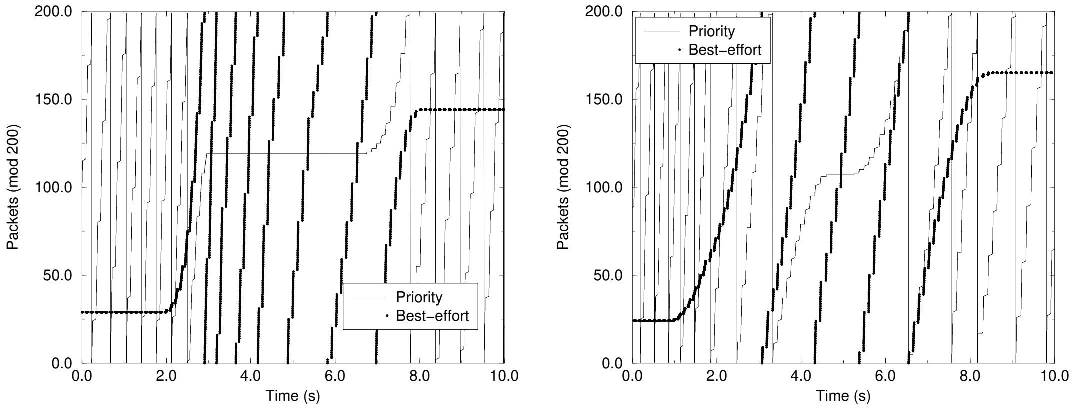
Figure 6.4: Effect of external packet marking.

network. Similarly, when packet marking is turned off, a spike of unmarked packets may be injected into the the network.

Figure 6.5(a) shows a sample packet trace of a connection using this algorithm. The figure plots the number of marked and unmarked packets sent. As the figure shows, as soon as the connection reaches its target, the PME quickly cuts down the number of marked packets sent and starts sending a large amount of unmarked packets. In the simulations performed, the impact from the bursts of marked and unmarked packets was relatively minor. This is due to the fact that the TCP congestion control algorithm controls the combined stream of marked and unmarked packets in a very network-friendly fashion. The use of a common queue for marked and unmarked packets also adds to the stability. Even when the PME changes P_{mark} in large steps, the overall impact is a mere replacement of marked packets by an equal number of unmarked packets or vice versa. However, in situations where not all of the sources use TCP or where not all queues are ERED queues, large swings in the number of marked and unmarked packets can potentially lead to network instability.

6.3.2 TCP-friendly marking

In order to minimize the chances of triggering such instability in the network, the PME should update marking probabilities in a manner that is more network-friendly, while maintaining the ability to react to the changes in network load. To address the potential shortcoming of the algorithm



(a) P_{mark} = 1.00 (b) TCP-friendly marking

Figure 6.5: Burstiness of packet marking schemes

presented in Figure 6.2, Figure 6.6 shows an algorithm that updates P_{mark} in a more network-friendly manner. This algorithm draws on the windowing mechanisms used in TCP and tries to ensure that the number of marked (or unmarked) packets in the network increases by no more than 1 per round-trip time. This is in some sense similar to the linear increase algorithm for congestion avoidance used by TCP [34]. As shown in Figure 6.6, an estimated number of marked packets in flight (PWND) is computed by taking the estimated congestion window given as the product of the observed bandwidth and the estimated round-trip time (RTT) and multiplying it by the marking probability. At every update epoch, if the observed bandwidth is less than the target rate, PWND is incremented linearly ($\frac{1}{\text{CWND}}$). This ensures that the number of marked packets increases by no more than one in every round-trip time. Similarly, when the observed bandwidth is higher than the target rate, the decrease in the number of marked packets (and hence increase in the number of unmarked packets) is limited to one every round-trip time. Figure 6.5(b) shows the packet trace of the modified scheme. Unlike the previous trace, this time the connection slowly increases and decreases the number of marked and unmarked packets sent. Figure 6.7(a) shows the result from the same experiment with the PME implementing the packet marking algorithm presented in Figure 6.6. As seen from the graph, the marking algorithm is very reactive to changes in the network load and hence observed throughput. Consequently, connection *C2* maintains an average throughput at or above its $4Mbs$ target most of the time. However, it changes P_{mark} in a more network-friendly fashion and reduces the risk of network instability.

Every acknowledgment:

PWND = $P_{mark} \times B_o \times RTT)$

if $(B_o < B_t)$

PWND = PWND $+ \frac{1}{\text{CWND}}$

else

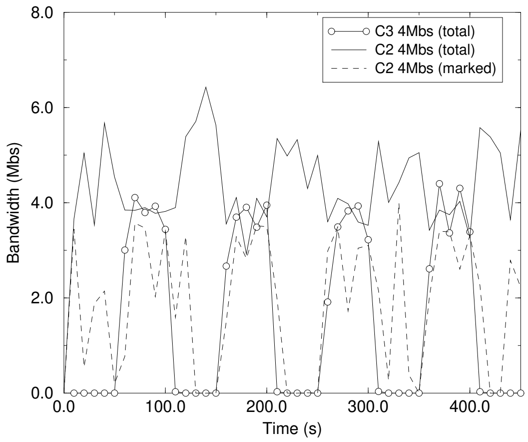
PWND = PWND $- \frac{1}{\text{CWND}}$

P_{mark} = $\frac{\text{PWND}}{B_o \times RTT}$

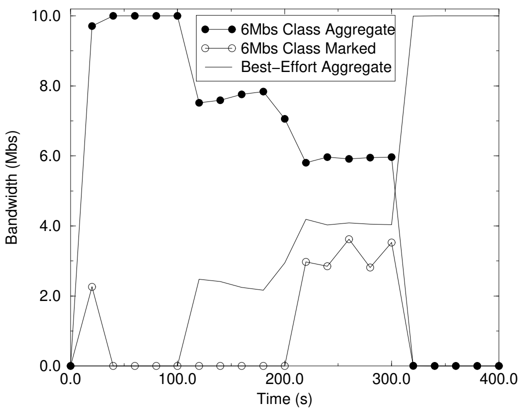
Figure 6.6: TCP-friendly algorithm for changing P_{mark}

6.3.3 Marking aggregates

While the previous experiments show how per-connection target throughputs can be achieved, PME can also meet the throughput target of an aggregation of connections. As in the case of individual connections, it simply monitors the throughput of the connection group and adjusts the marking rate based on the observed throughput and requested target. Figure 6.7(b) shows the results of an experiment where a PME controls two sets of connections sharing a $10Mbs$ bottleneck link. The first set of connections requires at least $6Mbs$ of bandwidth at all times while the other set is simply treated as best-effort. In this simulation, there are 3 identical connections in the first set and 4 identical connections in the second set. Initially, only the three connections of the first set are active. Thus, the aggregate bandwidth seen is the entire link bandwidth with each source receiving a third of the bandwidth. Note that the marking rate for the connection group is zero as there is enough bandwidth available to meet the target service level. At $t = 100s$, one best-effort connection is started. Since an even split of the bandwidth gives each connection approximately $2.5Mbs$, the three connections in the first set get a total of $7.5Mbs$ without any packet marking. At $t = 200s$, the other three best-effort connections are started. In this case, an even split of the bandwidth across all connections is not sufficient to sustain the target rate of $6Mbs$ for the first set. Thus, the PME begins to mark packets in order to sustain the target rate of $6Mbs$. As the figure shows, the marking increases to a level sufficient to maintain the target rate. The best-effort connections then get an equal share of the leftover $4Mbs$. Finally, at $t = 300s$, all connections of the first set are terminated. As the figure shows, the best-effort connections get the entire $10Mbs$ with each getting a fair share of it.



(a) Transient experiment



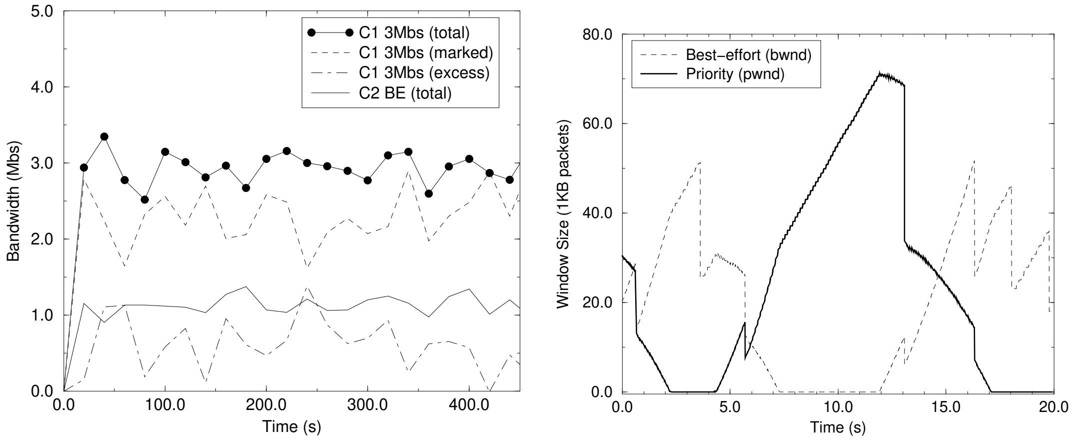
(b) Aggregation experiment

Figure 6.7: Performance of TCP-friendly algorithm.

6.4 Source Integrated Marking

One of the problems with having the PME external and transparent to the source is that it has little control on the flow and congestion control mechanisms exercised by the source. This lack of control can have detrimental impact on performance. For example, while a source-transparent PME is fairly effective in maintaining the observed throughput close to the target bandwidth, it often marks more packets than required. In an ideal scenario, a connection that stripes its packets across two priorities should receive a fair share of the best-effort bandwidth in addition to the bandwidth received due to priority packets. A TCP source oblivious of the packet marking fails to compete fairly with best-effort connections for its share of best-effort bandwidth. Consequently, the PME marks more packets than it should have, had the connection received its fair share of the best-effort bandwidth.

Figure 6.8(a) presents results from an experiment that demonstrates this. In this experiment, a connection *C1* with a target bandwidth of $3Mbs$, and 5 best-effort connections (*C2*, *C3*, *C4*, *C5*, *C6*) between nodes $n0$ and $n5$ are simulated. Figure 6.8 shows the marking rate, the best-effort bandwidth, and the total bandwidth received by *C1* along with the total bandwidth received by *C2*, one of the 5 identical best-effort connections. As shown in the figure, *C1* gets a much smaller share of the best-effort bandwidth than *C2*. Thus, it must mark a larger portion of its packets than it should in order to maintain the desired level of performance. This phenomenon can be easily explained if



(a) Bandwidth graph

(b) Window trace of $3Mbs$ connection

Figure 6.8: Bandwidth sharing using source-transparent marking

the window trace of the $3Mbs$ connection is examined. Figure 6.8(b) plots both the priority and best-effort portions of the connection's congestion window. As the figure shows, when the application requires additional bandwidth it must send priority packets *in place* of best-effort packets. Thus, when the connection sends priority packets, it cannot compete fairly for the available best-effort bandwidth.

In order to address this problem, a PME which is integrated with the TCP sender is considered. Figures 6.9 and 6.10 show the new algorithm. In this scheme, the congestion window (CWND) maintained by a TCP source is split into two parts: (1) a priority window (PWND) which is a measure of the number of marked packets that are in the network, and (2) a best-effort window (BWND) that reflects the number of unmarked packets that are outstanding. Upon a loss, the sender determines whether the lost packet was sent as a marked or an unmarked packet. The loss of a marked packet is an indication of severe congestion in the network. Consequently, both the priority and best-effort windows are reduced. However, the loss of an unmarked packet is an indication of congestion potentially only in the best-effort service class and hence only the best-effort window is reduced. The procedure for opening the congestion window is also modified. The connection keeps track of two additional thresholds values, namely PSSTHRESH and BSSTHRESH which are updated whenever the connection experiences a priority and a best-effort loss, respectively. When a connection is below its target bandwidth, it opens up both the priority and best-effort windows. If either one of the windows is below its respective threshold (PSSTHRESH and BSSTHRESH), it is in the slow

$$
\begin{array}{l}
\text{After every acknowledgment (opencwnd)} \\
\quad \text{PWND} = P_{mark} \times \text{CWND} \\
\quad \text{BWND} = (1\text{-}P_{mark}) \times \text{CWND} \\
\quad \textbf{if } (B_o < B_t) \\
\quad\quad \textbf{if } (\text{PWND} < \text{PSSTHRESH}) \\
\quad\quad\quad \text{PWND} = \text{PWND} + \frac{\text{PWND}}{\text{CWND}} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \text{PWND} = \text{PWND} + \frac{1}{\text{CWND}} \\
\quad\quad \textbf{if } (\text{BWND} < \text{BSSTHRESH}) \\
\quad\quad\quad \text{BWND} = \text{BWND} + \frac{\text{BWND}}{\text{CWND}} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \text{BWND} = \text{BWND} + \frac{1}{\text{CWND}} \\
\quad \textbf{else} \\
\quad\quad \textbf{if } (\text{PWND} > 0) \\
\quad\quad\quad \textbf{if } (\text{BWND} < \text{BSSTHRESH}) \\
\quad\quad\quad\quad \text{PWND} = \text{PWND} - \frac{\text{BWND}}{\text{CWND}} \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \text{PWND} = \text{PWND} - \frac{1}{\text{CWND}} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \textbf{if } (\text{BWND} < \text{BSSTHRESH}) \\
\quad\quad\quad\quad \text{BWND} = \text{BWND} + \frac{\text{BWND}}{\text{CWND}} \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \text{BWND} = \text{BWND} + \frac{1}{\text{CWND}} \\
\quad \text{if } (\text{PWND} < 0) \text{ PWND} = 0 \\
\quad \text{CWND} = \text{PWND} + \text{BWND} \\
\quad P_{mark} = \frac{\text{PWND}}{\text{CWND}}
\end{array}
$$

Figure 6.9: Customized TCP congestion window opening.

start mode. Note that the increases are scaled so that the overall congestion window does not grow any faster than that in an unmodified TCP. Scaling these increases is slightly conservative, since it temporarily hinders the source from growing its best-effort window as quickly as other best-effort sources. However, the conservative behavior aids in avoiding congestion collapse scenarios. When either window is above its threshold, it increases linearly (i.e., one segment per round-trip time). Note that while CWND grows by two segments every round-trip time, the best-effort part of the window (BWND) only grows as quickly as the CWND of a best-effort connection. While this modified windowing algorithm is essential in obtaining a fair share of the best-effort bandwidth in a network that supports service differentiation, it essentially behaves like two fairly independent connections. In a network that does not support end-to-end service differentiation, a TCP source modified in this manner may receive twice as much bandwidth as compared to unmodified TCP sources. Additional modifications to address this aspect are discussed in Section 6.5. Figure 6.11

After every segment loss from dupack (closecwnd) PWND = $P_{mark} \times$ CWND BWND = (1-P_{mark}) $\times$ CWND **if** (priority loss) CWND = $\frac{\text{CWND}}{2}$ PSSTHRESH=$P_{mark} \times$ CWND BSSTHRESH=(1-P_{mark}) $\times$ CWND **else** BWND = $\frac{\text{BWND}}{2}$ BSSTHRESH = BWND CWND = PWND + BWND P_{mark} = $\frac{\text{PWND}}{\text{CWND}}$

Figure 6.10: Customized TCP congestion window closing.

shows results from the experiment presented in Figure 6.8 using the algorithm described above. In contrast to Figure 6.8(a), the amount of best-effort bandwidth received by the $3Mbs$ source closely matches the bandwidth received by the best-effort sources. Figure 6.11(b) shows the priority and best-effort windows of the $3Mbs$ connection. In contrast to Figure 6.8(b), the connection is able to compete for best-effort bandwidth independent of the priority marking.

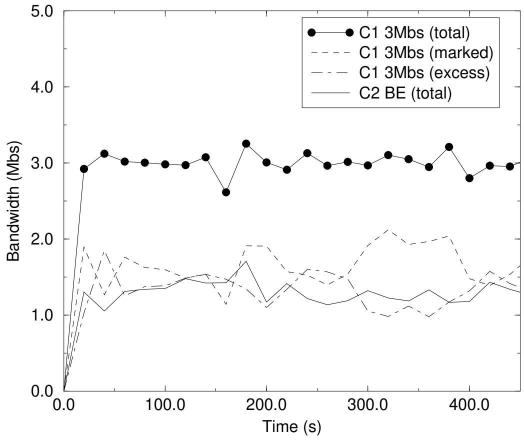
By taking a closer look at the packet marking rate and its deviation from the theoretically computed optimal marking rate, the issue of fair bandwidth sharing can be further examined[1]. The computation of ideal marking rates is quite straightforward. For example, consider a network with a bottleneck link of bandwidth B. Assume that n connections with target rates of R_i, $i = 1, 2, \ldots, n$, are passing through it. Let r_i be the optimal marking rate of the connection with a target rate of R_i, and let b be share of best-effort bandwidth received by all connections. A connection j with $R_j < b$, is essentially a best-effort connection with $r_j = 0$. The following set of equations capture the system constraints:

$$r_i + b = R_i$$

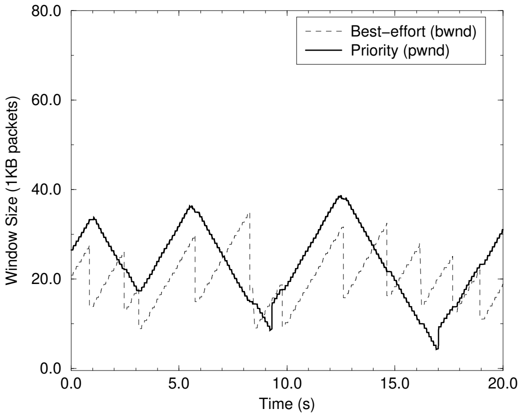
$$\sum_{i=1}^{n} r_i + nb = B$$

Figure 6.12 shows the results of an experiment with two connections *C1* and *C2* with target rates of $3Mbs$ and $2Mbs$, respectively, and six best-effort connections sharing a bottleneck link of $10Mbs$. The connections *C1* and *C2* start at time $t = 0s$, followed by two best-effort connections

[1]Note that when optimal marking is achieved, accurate congestion-based pricing can be done using the marking rate of a connection.



(a) Bandwidth graph

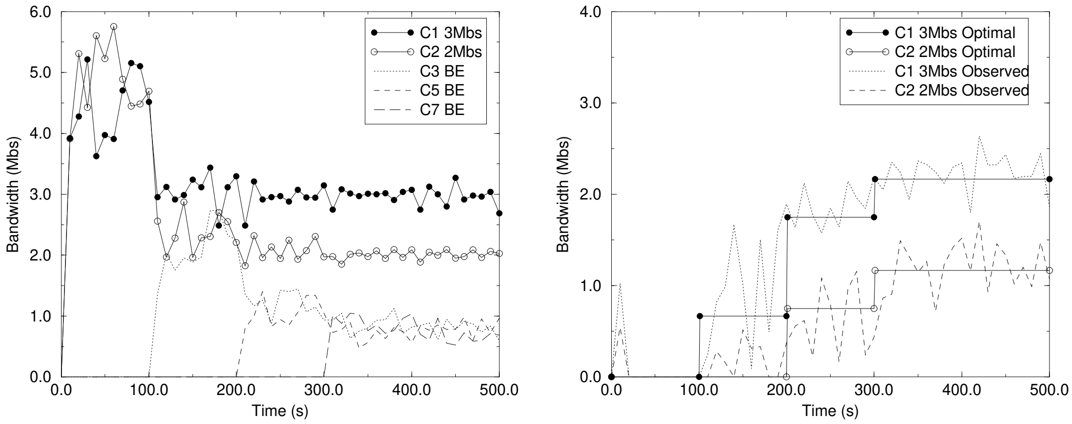


(b) Window trace of $3Mbs$ connection

Figure 6.11: Bandwidth sharing using source-integrated marking

at $t = 100s$, another two at $t = 200s$, and the last two at $t = 300s$. Figure 6.12(a) shows the bandwidth received by *C1* and *C2* and three of the best-effort connections. Figure 6.12(b) shows the marking rate of both *C1* and *C2*, as well as their calculated ideal marking rates. At time $t = 0s$, when only two connections are on-line, a fair split of the bandwidth satisfies target rates of both *C1* and *C2*. Thus, neither source marks any of their packets and each gets approximately half of the bottleneck bandwidth. At $t = 100s$, two best-effort connections are added. At this point, *C1* needs to mark at a $0.67Mbs$ rate and each of the sources should get $2.33Mbs$ of the excess best-effort bandwidth. Since *C2*'s share of best-effort bandwidth is more than its target rate, it need not mark any of its packets. As Figure 6.12 shows, the marking rate and total bandwidth graphs reflect the change. At $t = 200s$, two more best-effort connections are added. Now, *C1* has to mark at a rate of $1.75Mbs$ while *C2* needs to mark at at a rate of $0.75Mbs$. This leaves each source $1.25Mbs$ of the excess bandwidth. As the total bandwidth graph shows, the best-effort connections get about $1.25Mbs$ while *C1* and *C2* get their respective target bandwidths. The marking rates of *C1* and *C2* also adapt to this change, increasing to the optimal marking rates. Finally, at $t = 300s$, the last two best-effort sources are added. This time, *C1* needs to mark at $2.17Mbs$ while *C2* needs to mark at $1.17Mbs$. Each connection now gets $0.83Mbs$ of the excess bandwidth. Again, as the graphs show, both the priority and best-effort connections perform as expected.

To examine the impact that the windowing modifications have, the same set of experiments with a source-transparent PME was performed. Figure 6.13 shows the total bandwidth and marking rate



(a) Total bandwidths

(b) Marking rates

Figure 6.12: TCP with integrated packet marking

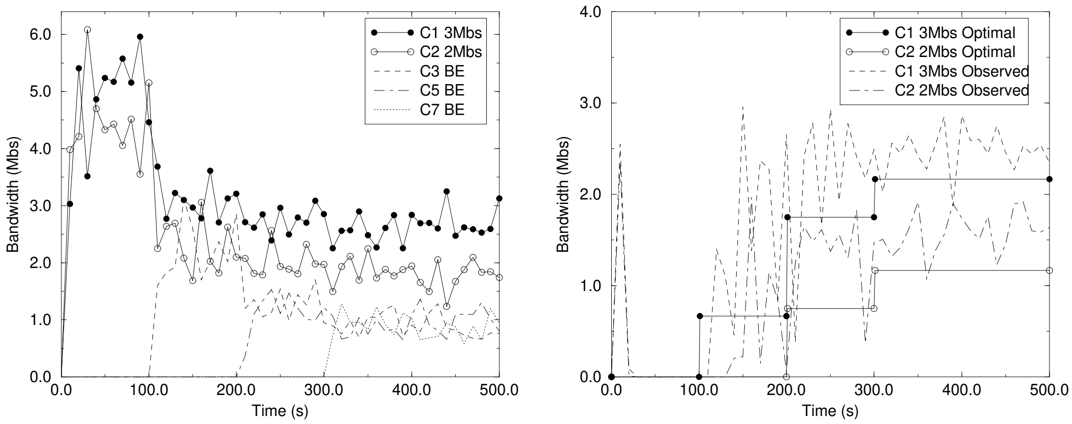
for different connections. Since TCP windowing restricts connections *C1* and *C2* from competing for the excess bandwidth, the PME consistently overmarks its packets as shown in Figure 6.13(b). Increased marking can potentially fill the ERED queue with marked packets, making it behave more like a regular RED queue. As Figure 6.13(a) shows, loss of priority packets causes periods of time where throughputs of connections *C1* and *C2* drop significantly below their target rates.

6.5 Deployment Issues

The Internet is a conglomeration of a large number of heterogeneous devices. Because of this, deployment of any proposed architecture is difficult. In this section, a number of important deployment issues are addressed. In particular, the performance of the proposed architecture in oversubscribed situations, in the presence of non-responsive flows, and in a network of heterogeneous routers and sources is considered.

6.5.1 Handling oversubscription

One of the key advantages of using an adaptive packet marking scheme is that it obviates the need for a signaling protocol. However, since there is no resource reservation, the service guarantees it provides are necessarily soft. In an RSVP-based architecture, when demand for service continually exceeds the capacity, admission control is used to deny additional connections access to the network



(a) Total bandwidths

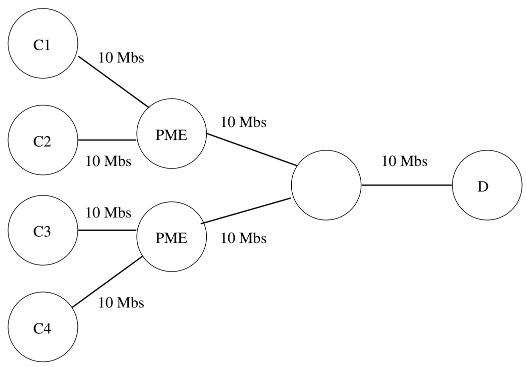
(b) Marking rates

Figure 6.13: TCP with transparent packet marking

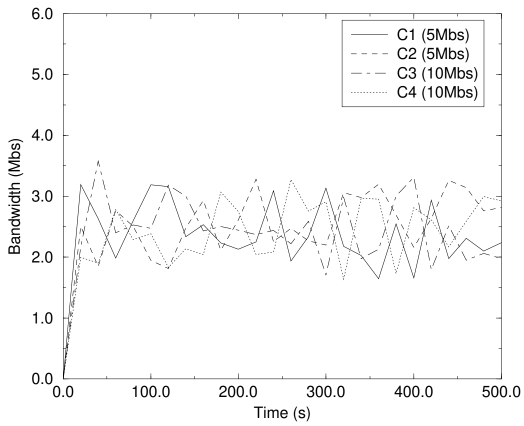
in order to maintain the service levels of the current set of connections. In networks where no reservations or admission control is in place, the network must instead offer degraded service at times of overload. In both cases, pricing and access policies in conjunction with capacity planning must be used to balance the supply and the demand of network resources. This section describes how oversubscription is handled in the proposed service model.

When aggregate demand exceeds capacity, all connections with non-zero target rates carry only marked packets. Consequently, they only compete for priority bandwidth and the ERED queue at the bottleneck degenerates into to RED queue serving only priority traffic. In the case of a source-transparent PME, since the underlying TCP windowing algorithm is not changed, the requested target bandwidth does not influence the throughput a source receives. Consequently, each source receives an equal fair share of the bottleneck bandwidth.

Oversubscription results in the same outcome when the PME is integrated within the source. In this case, since the algorithms for growing and shrinking the priority window are independent of the bandwidth demand, the windowing algorithm simply behaves as normal TCP. This adaptation in presence of overload prevents possible congestion collapse. Figure 6.14(a) shows an example scenario with four connections *C1*, *C2*, *C3*, and *C4* spanning the network. The connections *C1* and *C2* have a target rate of $5Mbs$ each while connections *C3* and *C4* aim at a target rate of $10Mbs$. As the figure shows, by using the integrated marking scheme, each connection gets a fair share of the bottleneck bandwidth when the demand exceeds the capacity.



(a) Network configuration



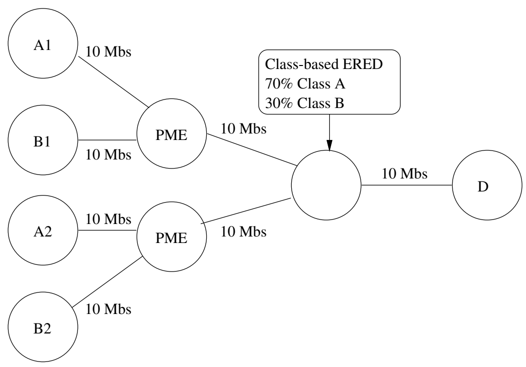
(b) Bandwidth graph

Figure 6.14: Oversubscription

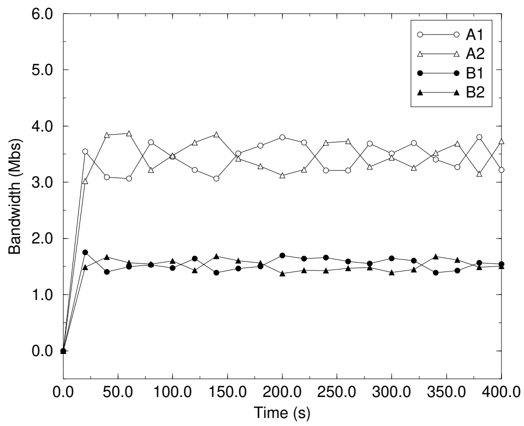
Another approach to handle oversubscribscription is to provide weighted bandwidth sharing depending on the target rates or the importance of the connections or connection groups. Since the proposed scheme uses only a single priority bit, it cannot itself be used to provide weighted bandwidth sharing in times of oversubscription. However, it is possible to implement weighted bandwidth sharing by using additional priority levels which give the network an indication of the connection's target rate and/or importance. For example, consider a more elaborate service architecture where additional priority bits are used to direct traffic into different ERED queues. These queues are then served using any one of various proposed queueing disciplines, such as weighted-fair queueing, class-based queueing, or even strict priority queueing. Figure 6.15(a) shows an example scenario in which an additional bit is used to select separate queues in a class-based queue [27]. In this example, the class-based queue is configured to provide applications and/or hosts in one class (A) with at least 70 percent of the allocated bandwidth. The remaining 30 percent of the bandwidth is allocated to the other class (B). When the applications in class A (A1 and A2) and class B (B1 and B2) request more bandwidth than is available, the additional priority encoding allows the network to maintain weighted bandwidth sharing between the two classes as shown in Figure 6.15(b).

6.5.2 Dealing with non-responsive flows

One of the potential risks in an adaptive approach to service differentiation is that proliferation of applications which do not adapt to network dynamics can lead to severe performance degradation



(a) Network configuration



(b) Bandwidth graph

Figure 6.15: Oversubscription and multiple priorities

and even congestion collapse. Thus, an important issue in deploying the proposed scheme is the protection of the network against non-responsive flows [16, 41]. A salient feature of this scheme is that it provides performance incentives for applications to adapt to network dynamics and help avoid congestion collapse. When used in conjunction with intelligent queue management mechanisms, it can also penalize non-responsive flows.

Figure 6.16 shows a network configuration which consists of four TCP connections ($T1$, $T2$, $T3$, and $T4$) which are competing for bandwidth with a non-responsive flow ($M1$) across a $10Mbs$ link. The aggregate target rate for the TCP connections is $7Mbs$. The target rate for the non-responsive flow is $3Mbs$. Initially, only the TCP sources are active and each competes fairly for the link bandwidth. The non-responsive flow starts transmitting at $1Mbs$ at $t = 100s$, and at $3Mbs$ at $t = 200s$. As shown in the figure, the aggregate throughput of the TCP connections drops when the non-responsive flow becomes active, but remains at a rate close to $7Mbs$. At $t = 300s$, the non-responsive flow increases its transmission rate to $5Mbs$, thus exceeding its allocated rate of $3Mbs$. As shown in the figure, the marking rate of this flow immediately drops to zero and the loss rate increases to approximately the difference between the transmission rate and the allocated rate. The reason why this happens is that the PME observes that the non-responsive flow is sending packets at a rate which is higher than its given rate. In order to encourage sources to send packets which are deliverable, the PME counts every packet it receives for a particular flow against its allocation. The non-responsive flow further increases its transmission rate to $7Mbs$ at $t = 400s$. Again, the

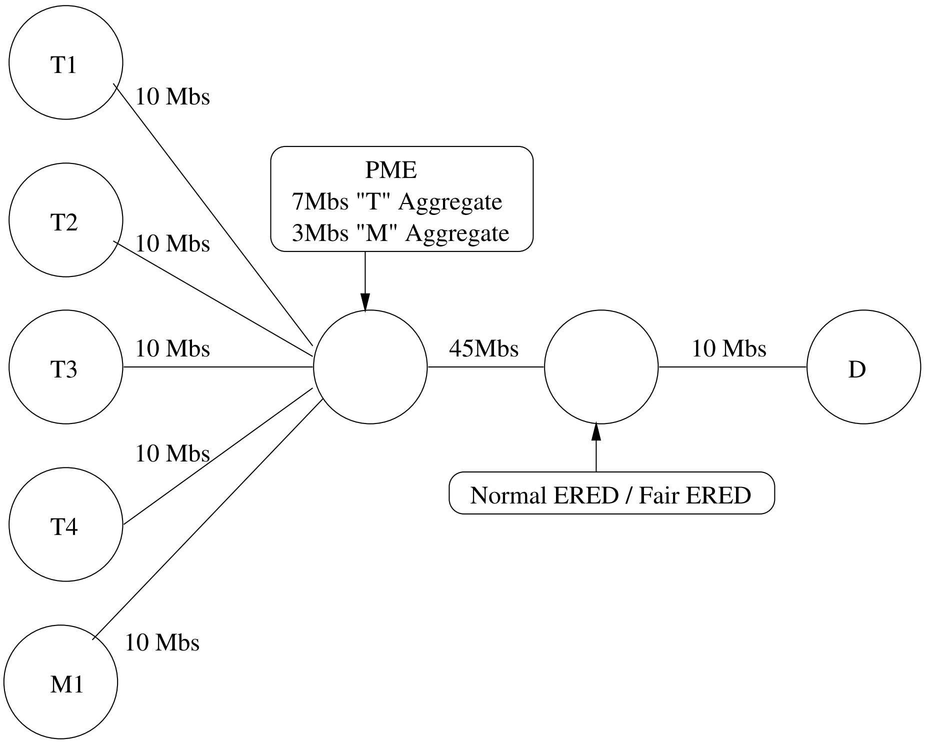
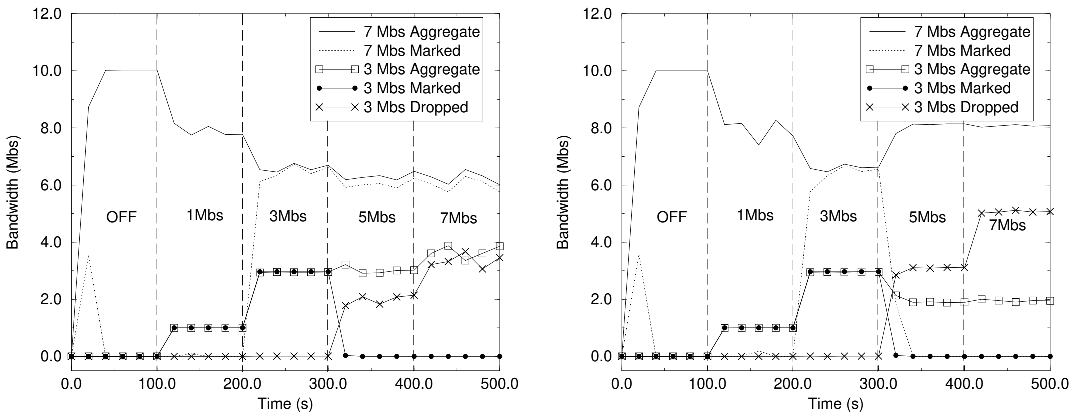


Figure 6.16: Network topology

throughput observed by the flow remains fairly constant near its allocated rate of $3Mbs$, while the amount of packets which are dropped increases at the same rate as the transmission rate. Thus, the non-responsive flow gains little by transmitting any amount above its allocated rate.

In the previous experiment, the non-responsive flow does, in fact, have a negative impact on the TCP connections. As Figure 6.17(a) shows, the aggregate marking rate of the TCP connections approaches the aggregate transmission rate, since the unmarked packets from the non-responsive flow dominates any of the excess bandwidth available. In effect, the non-responsive flow obtains all of the available best-effort bandwidth while shutting out all other well-behaved connections. In order to provide better fairness between connections competing for best-effort bandwidth, the bottleneck ERED queue is enhanced with additional fairness mechanisms based on FRED [41]. Figure 6.17(b) shows the results of the experiment. As the figure shows, when the non-responsive flow begins transmitting at a rate higher than $3Mbs$, the PME reduces its marking to zero as described earlier. Since the flow does not respond to congestion signals given by the bottleneck queue and continues to send an inordinate amount of unmarked packets, the fair ERED queue detects the flow and limits



(a) Normal ERED queue (b) Fair ERED queue

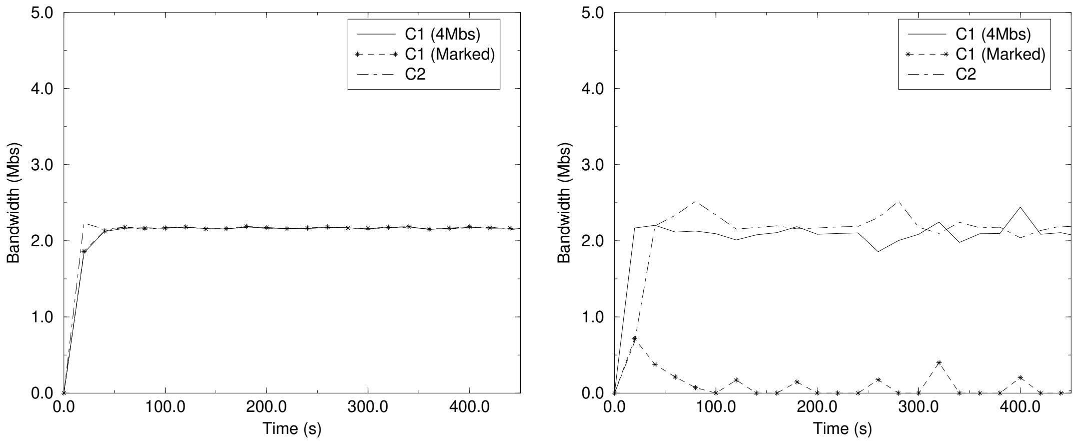
Figure 6.17: Non-responsive flows

its throughput to a fair share of the best-effort bandwidth. In this case, a fair share of the bandwidth is $2Mbs$. Thus, by sending over its target rate of $3Mbs$ without regard to congestion in the network, the non-responsive flow reduces its own observed throughput to $2Mbs$. Note that given a fair share of the best-effort bandwidth, the TCP flows can now maintain their $7Mbs$ aggregate target rate without marking any packets. This is in contrast to Figure 6.17(a), where the TCP flows are forced to have all of their packets marked in order to maintain their target rate. Thus, the non-responsive flow hurts itself while helping other flows as it sends over its target rate without regard to network congestion.

6.5.3 Handling heterogeneity

The Internet is a highly heterogeneous and slowly evolving networking environment. It is impractical to assume that all routers in the Internet will handle priority packets in the same way. As a matter of fact, it is quite likely that only a fraction of them will support service differentiation between packets of different priorities. In order to be successful in this environment, it is important that any packet marking scheme proposed is capable of handling heterogeneity in the network. More specifically, it should be able to operate in an environment where all routers do not support service differentiation between priority and best-effort packets.

One of the salient features of the proposed scheme is its ability to operate in a network that does not provide service differentiation. When the PME is transparent to the source, TCP transmis-

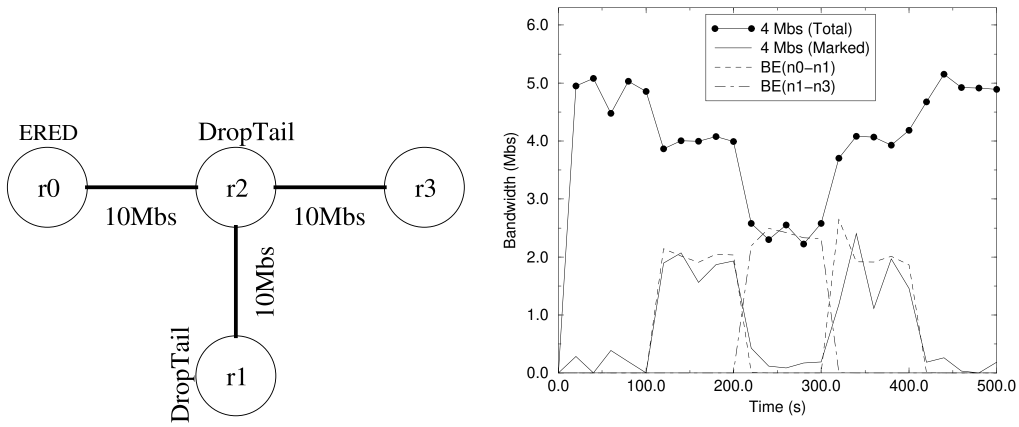


(a) Packet marking gateway (b) TCP with windowing modifications

Figure 6.18: Performance over an all drop-tail network

sion control mechanisms are not affected as a result of packet marking. Thus, the lack of service differentiation simply makes the packet marking ineffective and the TCP sources behave as if they are operating in a best-effort network. When the PME is integrated with the source, the situation is little different. In this case, there are essentially two connections with differing priorities. Hence, in absence of service differentiation, this scheme can potentially be twice as aggressive as a regular TCP connection. While such behavior may be justified when a user is charged for marked packets, it may be desirable to turn off marking when service differentiation is not supported by the network.

To address this, a simple mechanism for turning off the marking and modified windowing when the network does not support end-to-end service differentiation is implemented. Note that the bottleneck of a connection may shift from a link that supports service differentiation to one that does not, and vice versa. Hence detection of service differentiation on a connection path is not a one-time process; it requires constant monitoring. To minimize the cost of monitoring and, at the same time, remain reactive to changes in the network dynamics, an exponential back-off algorithm is used to determine monitoring intervals. In particular, the source keeps track of the inter-drop times for both priority and best-effort packets. In a network which supports service discrimination, the number of priority packets transmitted between successive priority packet drops is expected to be substantially greater than the number of best-effort packets transmitted between successive non-priority packet drops. When this is not the case, the source simply turns off the marking and the windowing algorithm, reverting back to normal TCP. After a preset interval, marking is turned on again and



(a) Network (b) Bandwidth graph

Figure 6.19: Effects of heterogeneity

the source monitors inter-drop intervals to detect service differentiation. If it fails to detect service differentiation, it shuts down marking for twice the duration it had before. If the source observes that service differentiation is supported by the network, the connection continues using the modified windowing algorithm and resets the back-off interval to its initial (smaller) value.

Figure 6.18(a) shows the throughput observed by five connections *C1*, *C2*, *C3*, *C4*, and *C5* going from *n0* to *n5* when all of the queues in the network are drop-tail queues with no support for service differentiation. Connection *C1* has a target rate of $4Mbs$. All other connections are best-effort. A source-transparent PME is used to mark packets in this example. As expected, bottleneck bandwidth is shared fairly among all five connections. Note that the packets are continually being marked even though the network does not honor their DS marking. This is because the PME cannot determine that the DS field in the packets is being ignored unless it keeps additional information. Since the connection is always below its target bandwidth, the PME simply marks all of its packets. Figure 6.18(b) shows the same experimental setup as before. However, in this example, the PME is integrated within the source. As the figure shows, *C1* backs off its marking as it detects that the network does not support any service differentiation. Thus, the connection competes fairly with all of the other best-effort connections for the excess bandwidth.

The back-off mechanisms used when the PME is integrated into the source adapt quickly to the changes in the network. This helps the source adapt its windowing and marking strategy as the bottleneck link shifts from non-priority to priority queues in a heterogeneous network. Figure 6.19(a) shows a network with 4 nodes where *r0* implements the ERED queueing mechanism while *r1* and *r2* are simple drop-tail gateways. In this network, two priority connections *C1* and *C2* with $4Mbs$ target bandwidths and several transient best-effort connections are simulated. The transient connections are used to move the bottleneck link from *r0-r2* to *r2-r3*. Figure 6.19(b) shows the throughputs seen by different sources as the bottleneck moves from one link to another. Initially, connections *C1* and *C2* going from *r0* to *r3* are active. In the absence of any other connections, neither connection needs to mark any of their packets in order to achieve their target rates. At $t = 100s$, a best-effort connection is spawned between *r0* and *r1*. Since a fair share of the bottleneck bandwidth of $10Mbs$ does not satisfy the target rates of connections *C1* and *C2*, they both mark their packets at a rate of $2Mbs$. From the equations outlined in Section 6.4, this is the optimal marking rate in this scenario. Each connection also receives $2Mbs$ of the leftover best-effort bandwidth. At $t = 200s$, the best-effort connection terminates and two new best-effort connections are started between nodes *r1* and *r3*. At this time, the bottleneck link is between *r2* and *r3* which happens to be a drop-tail queue with no support for service differentiation. In this case, even though *C1* and *C2* fail to sustain their target rates, they back off their marking and revert back to the original windowing algorithm. Consequently, all four connections now receive an equal share of the bottleneck bandwidth of $10Mbs$. At $t = 300s$, the best-effort connections terminate and a new best-effort connection is spawned between nodes *r0* and *r1*. At this point, the bottleneck shifts to the link *r0-r2* which supports service differentiation. This change is detected by *C1* and *C2* and they turn on marking to reach their target rate of $4Mbs$. Finally, at $t = 400s$, the best-effort connection terminates, leaving the network in its initial state. The connections *C1* and *C2* once again turn off their marking since they can support their target throughput without packet marking.

6.6 Conclusions

In this chapter, a number of adaptive packet marking algorithms for providing soft bandwidth guarantees over the Internet have been proposed and analyzed. Marking algorithms that are external and transparent to the source and algorithms that are integrated with the congestion and flow control mechanisms at the source have been examined. Both sets of algorithms have advantages and disadvantages from the standpoint of performance and deployment issues. The results presented in this chapter clearly demonstrate that simple service differentiation, and when used in conjunction with adaptive source control, can be an effective means to provide quality of service in the Internet.

This work can be extended in several ways. For example, the impact of marking packets at multiple places in the network is being investigated. Also under investigation is the interaction and interoperability of the proposed schemes with alternative mechanisms to support quality of service in the Internet. As described in Chapter 5, enhancements to BLUE for performing service differentiation are also being considered. Finally, generalization of the two priority scheme to multiple priorities is being examined.

CHAPTER 7

CONCLUSION

This thesis has focused on solving two extremely important challenges to today's Internet: supporting an explosion in the number of users and supporting a myriad of new applications which require more out of the network than the best-effort service that the Internet currently provides. To this end, a number of modifications to the basic congestion control and queue management algorithms of the Internet have been examined. More specifically, this thesis has:

1. Shown that even with ECN, current active queue management mechanisms such as RED are ineffective because they are not sensitive to the level of congestion in the network. To address this problem, an adaptive modification to RED which allows it to manage congestion more effectively has been developed, implemented and evaluated. In addition, this thesis has demonstrated the inherent weakness in all current active queue management mechanisms in that they rely on queue lengths to do congestion control. To address this problem, BLUE, a queue management algorithm based on the history of queue behavior has been developed, implemented and evaluated. BLUE outperforms all current active queue management algorithms by a large margin in terms of packet loss rates and buffer space requirements.

2. Analyzed TCP in order to show weaknesses in its congestion control algorithms when a large number of connections are present. To address this, a modification to TCP congestion control which allows sources to fully utilize network capacity without packet loss in the presence of an arbitrarily large number of connections has been developed and evaluated. When used in conjunction with adaptive queue management mechanisms, these modifications have been shown to maximize network efficiency even with an extremely limited amount of network

buffers.

3. Developed and analyzed SFB, a novel mechanism for scalably enforcing fairness between a large number of connections using a very small amount of buffer space and state.

4. Developed and analyzed a new form of providing QoS over the Internet based on priority marking. This was one of the first pieces of work in differentiated services (Summer 1996) and eventually led to the formation of the IETF's DIFFSERV working group in early 1998. This work addresses the problem of having disjunct rate control mechanisms in the network: TCP and priority marking. In addition, it has shown how such marking can be ineffective due to TCP dynamics. To address this problem, a number of modifications to TCP which allow TCP to take advantage of the priority mechanisms in the network have been proposed and evaluated.

5. Developed an architecture for providing soft bandwidth guarantees in an easily deployable manner. As part of this architecture, a number of novel mechanisms for integrating packet marking into end hosts has been proposed and evaluated. These mechanisms allow optimal marking rates to be obtained between sources and allow hosts to scalably detect heterogeneity and lack of service differentiation in the network.

With the rapid increase in users and applications caused by the success of the WWW, it is imperative that the infrastructure in place in the Internet be able to meet these new challenges. While this thesis has examined a large number of issues and has provided a number of solutions, there are still several open issues which need to be addressed. As part of on-going work, several key aspects of this dissertation are being extended. They include:

1. Parameterizing Adaptive RED and BLUE to current Internet traffic. By understanding how Internet traffic changes over a range of time scales, the performance of both queue management schemes can be optimized. In addition, a number of ways to predictively and pre-emptively set the parameters of Adaptive RED and BLUE are being considered. Since routers often have access to more detailed information on the number of flows which are currently active, it is possible to make these queue management algorithms even more proactive.

2. Developing more protective and adaptive queue management algorithms for supporting differentiated services. In particular, techniques for making BLUE priority-aware are currently being examined. By using BLUE queue management instead of RED, differentiated services can be supported with a minimal amount of buffer space.

3. Extending priority-aware congestion control beyond TCP by using RTP over UDP. While a large number of applications use TCP, the growing number of applications, and in particular, streaming audio and video applications do not.

4. Developing additional mechanisms for detecting whether or not service differentiation is being done on packets of a flow. Given the diversity of the Internet infrastructure, the ability to detect service differentiation is essential for both service providers and end users.

Given this additional work and the modifications described in this thesis, it is possible to significantly improve network efficiency in times of heavy congestion as well as provide a variety of predictable services to emerging applications. By maximizing network efficiency across a large range of loads, congestion collapse can be effectively prevented and the lifetime of many of the network links and routers currently in place can be extended. By providing mechanisms for scalably supporting service differentiation between flows in the Internet, the widespread deployment of a large class of bandwidth-sensitive applications can be enabled. Together, they will allow the Internet to continue to provide robust services to its users and applications well into the next century.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. *RFC 2414*, September 1998.

[2] P. Almquist. Type of Service in the Internet Protocol Suite. *RFC 1349*, July 1992.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *RFC 2475*, December 1998.

[4] 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. *RFC 2309*, April 1998.

[5] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: An Overview. *RFC 1633*, June 1994.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. *RFC 2205*, September 1997.

[7] 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.

[8] 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.

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

[10] I. Cidon, R. Guerin, and A. Khamisy. Protective Buffer Management Policies. *IEEE/ACM Transactions on Networking*, 2(3), June 1994.

[11] D. Clark. A Model for Cost Allocation and Pricing in the Internet. *MIT Workshop on Internet Economics*, March 1995.

[12] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proc. of ACM SIGCOMM*, pages 14–26, August 1992.

[13] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. *RFC 2460*, December 1998.

[14] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of Fair Queuing Algorithm. In *Proceedings of SIGCOMM*, 1989.

[15] S. Doran. RED Experience and Differentiated Queueing. In *NANOG Meeting*, June 1998.

[16] K. Fall and S. Floyd. Router Mechanisms to Support End-to-End Congestion Control. ftp://ftp.ee.lbl.gov/papers/collapse.ps, February 1997.

[17] W. Feng, D. Kandlur, D. Saha, and K. Shin. TCP Enhancements for an Integrated Services Internet. IBM Research Report RC 20618 and UM CSE-TR-312-96, November 1996.

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

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

[20] W. Feng, D. Kandlur, D. Saha, and K. Shin. Adaptive Packet Marking for Providing Differentiated Services in the Internet. In *Proc. of ICNP '98*, October 1998.

[21] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. In *Proc. IEEE INFOCOM*, March 1999.

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

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

[24] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. *Internet Draft draft-ietf-tcpimpl-newreno-02.txt*, February 1999.

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

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

[27] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[28] S. Golestani. A Self-Clocked Fair Queuing Scheme for Broadband Applications. In *Proceedings of INFOCOM*, June 1994.

[29] P. Goyal, H. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM*, pages 157–168, August 1996.

[30] R. Guerin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Through Buffer Management. In *Proceedings of ACM SIGCOMM*, September 1998.

[31] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *Internet Draft draft-ietf-diffserv-af-04.txt*, January 1999.

[32] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of ACM SIGCOMM*, pages 270–280, August 1996.

[33] IEEE 802.11 Working Group. IEEE 802.11 Standard, June 1997.

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

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

[36] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. *RFC 1323*, May 1992.

[37] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. *Internet Draft draft-ietf-diffserv-phb-ef-01.txt*, November 1998.

[38] 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*, September 1984.

[39] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks. In *Proc. of ACM SIGCOMM*, pages 2–13, August 1995.

[40] 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.

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

[42] 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.

[43] J. MacKie-Mason and H. Varian. Pricing the Internet. In *Public Access to the Internet*, pages 269–314, May 1995.

[44] M. Mathis and J. Mahdavi. Forward Acknowledgement: Refining TCP Congestion Control. In *Proceedings of ACM SIGCOMM*, pages 281–291, August 1996.

[45] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. *RFC 2018*, October 1996.

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

[47] P. McKenney. Stochastic Fairness Queueing. In *Proc. IEEE INFOCOM*, March 1990.

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

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

[50] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. *RFC 2474*, December 1998.

[51] 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.

[52] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of ACM SIGCOMM*, September 1998.

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

[54] J. Postel. Internet Protocol. *RFC 791*, September 1981.

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

[56] E. Rathgeb. Modeling and Performance Comparison of Policing Mechanisms for ATM Networks. *IEEE Journal on Selected Areas of Communication*, 9(3), 1991.

[57] J. Rexford, A. Greenberg, and F. Bonomi. Hardware-Efficient Fair Queueing Architecture for High-Speed Networks. In *Proceedings of INFOCOM*, March 1996.

[58] J. Reynolds and J. Postel. Assigned Numbers. *RFC 1340*, July 1992.

[59] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. *RFC 2212*, September 1997.

[60] M. Shreedhar and G. Varghese. Efficient Fair Queuing using Deficit Round Robin. *IEEE/ACM Transactions on Networking*, 4(3), June 1996.

[61] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *RFC 2001*, January 1997.

[62] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM*, September 1998.

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

[64] V. K. Garg and K. Smolik and J. E. Wilkes. Applications Of CDMA In Wireless/Personal Communications. Prentice Hall Professional Technical Reference, October 1996.

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

[66] 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.

[67] 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.

[68] J. Wroclawski. Specification of Controlled-Load Network Element Service. *RFC 2211*, September 1997.

[69] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, pages 8–18, September 1993.

[70] 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.