Lightweight Active Router-Queue Management for Multimedia Networking

Mark Parris   *Kevin Jeffay*   *F. Donelson Smith*
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA
{parris,jeffay,smithfd}@cs.unc.edu
http://www.cs.unc.edu/Research/dirt

ABSTRACT

The Internet research community is promoting active queue management in routers as a proactive means of addressing congestion in the Internet. Active queue management mechanisms such as Random Early Detection (RED) work well for TCP flows but can fail in the presence of unresponsive UDP flows. Recent proposals extend RED to strongly favor TCP and TCP-like flows and to actively penalize "misbehaving" flows. This is problematic for multimedia flows that, although potentially well-behaved, do not, or can not, satisfy the definition of a TCP-like flow. In this paper we investigate an extension to RED active queue management called Class-Based Thresholds (CBT). The goal of CBT is to reduce congestion in routers and to protect TCP from all UDP flows while also ensuring acceptable throughput and latency for well-behaved UDP flows. CBT attempts to realize a "better than best effort" service for well-behaved multimedia flows that is comparable to that achieved by a packet or link scheduling discipline, however, CBT does this by queue management rather than by scheduling. We present results of experiments comparing our mechanisms to plain RED and to FRED, a variant of RED designed to ensure fair allocation of bandwidth amongst flows. We also compare CBT to a packet scheduling scheme. The experiments show that CBT (1) realizes protection for TCP, and (2) provides throughput and end-to-end latency for tagged UDP flows, that is better than that under FRED and RED and comparable to that achieved by packet scheduling. Moreover CBT is a lighter-weight mechanism than FRED in terms of its state requirements and implementation complexity.

Keywords: active queue management, multimedia networking, RED, congestion control.

1 INTRODUCTION

As the volume of traffic, and number of simultaneously active flows on Internet backbones increases, the problem of recognizing and addressing congestion within the network becomes increasingly important. There are two major approaches to managing congestion. One is to manage bandwidth through explicit resource reservation and allocation mechanisms such packet or link scheduling. This approach offers the potential of performance guarantees for classes of traffic but the algorithmic complexity and state requirements of scheduling makes its deployment difficult. The other approach is based on management of the queue of outbound packets for a particular link. This latter approach has recently become the subject of much interest within the Internet research community. For example, there is an increasing focus on the problem of recognizing and accommodating "well-behaved" flows — flows that respond to congestion by reducing the load they place on the network.[1] Both Braden *et al*., and Floyd *et al*., recognize TCP flows with correct congestion avoidance implementations as being well behaved and argue that these flows, as "good network citizens," should be protected and isolated from the effects of "misbehaving" flows [1, 2, 8, 11]. Examples of misbehaving flows include non-standard implementations of TCP, UDP connections that do not respond to indications of congestion, and UDP connections that are responsive to congestion but respond in ways other than those specified for TCP. A recent Internet draft considers the

* Supported by grants from the National Science Foundation (grants CDA-9624662, CCR-9510156, & IRIS-9508514), the Advanced Research Projects Agency (grant number 96-06580), the IBM Corporation, and a graduate fellowship from the Intel Corporation.

[1] We use the term *flow* simply as a convenient way to designate a sequence of packets having a common addressing 5-tuple of: *source and destination IP addresses*, *source and destination port numbers*, and *IP protocol type*.

In: *Multimedia Computing and Networking 1999*,
Proceedings, SPIE Proceedings Series, Volume 3654, San Jose, CA, January 1999, pages 162-174.

problem of congestion in the current Internet and makes two recommendations [2]. First, the authors recommend deploying active queue management schemes, specifically *Random Early Detection* (RED) to more effectively notify responsive flows of congestion [5]. Active queue management refers to extending the packet queueing discipline in the router beyond the commonly employed FIFO enqueue and dequeue policies. For example, under RED a router does not wait until the queue is full to drop packets. Instead, it probabilistically drops incoming packets when the queue's average length exceeds a threshold and automatically drops a random packet from the queue when the average exceeds a higher threshold. This provides earlier feedback, before the queue overflows, and probabilistically causes higher bandwidth flows to see a greater number of drops.

Second, Braden *et al*. recommend continued development of mechanisms to deal with flows that do not recognize packet loss as an indicator of congestion and respond to loss according to TCP's back-off algorithm. Such flows are problematic because they can, in the worst case, force TCP connections to transmit at their minimal possible rates while the unresponsive flows monopolize network resources. To date the problem of dealing with unresponsive/misbehaving flows has centered on how to constrain or penalize these flows [8, 13]. We recognize the need to protect well-behaved flows but also recognize that many applications choose unresponsive transport protocols, such as UDP, because they are concerned with throughput and (especially) latency rather than reliable delivery. Since reliable delivery in TCP depends on feedback, timeouts, and retransmissions, it can be incompatible with performance goals. Interactive multimedia applications are a prime example of applications that avoid TCP for performance reasons. These applications often use UDP instead of TCP because they are willing to trade low latency for unreliable delivery. Simply penalizing these UDP flows leaves application developers with some unattractive options. With the deployment of RED and its variants in many routers, application developers must realize that UDP flows will be subject to more aggressive drop policies than in the past. The developer could use TCP and incur overhead for features she may not want. Or, she could use another protocol and be subject to aggressive drop policies. Another alternative would be to use a protocol that implements TCP-like congestion control without the other undesired features such as reliable delivery [3].

We are investigating a different approach: the development of active queue management policies that attempt to balance the performance requirements of continuous media applications that use UDP with the need to both provide early notification of congestion to TCP connections and to protect TCP connections from unresponsive UDP flows. Specifically, we are experimenting with extensions to the RED queue management scheme for providing better performance for UDP flows without sacrificing performance for TCP flows. The key to our approach is to dynamically reserve a small fraction of a router's storage capacity for packets from well-behaved UDP connections (*e.g*., connections that employ application-level congestion control and avoidance mechanisms). Thus independent of the level of TCP traffic, a configurable number of tagged UDP connections are guaranteed to be able to transmit packets at a minimum rate. The goals of our approach, termed *Class-Based Thresholds* (*CBT*), are similar to other schemes for realizing "better-than-best-effort" service within IP, including packet scheduling and prioritization schemes such as *Class-Based Queuing* [7]. While we recognize packet scheduling techniques such as CBQ as the standard by which to measure resource allocation approaches, we are interested in determining how close we can come to the performance of these approaches using thresholds on a FIFO queue rather than scheduling. Moreover, we believe our design, using a queue management approach to be simpler and more efficient than these other schemes.

The following sections first describe other Active Queue Management schemes: RED, *Flow Random Early Detection* (FRED), and a packet scheduling scheme, CBQ. Section 3 briefly outlines the design of our CBT extension to RED. Section 4 then empirically compares CBT, FRED, RED, and CBQ and demonstrates that:

- CBT provides protection for TCP from unresponsive UDP flows that is comparable to that provided under FRED,
- CBT provides better throughput for tagged UDP flows than under RED and FRED,
- CBT results in tagged UDP packets transiting a router with lower latency than under FRED or RED, and
- CBT offers performance approaching that of CBQ.

Section 5 presents CBQ and argues that CBT is a simpler mechanism than either FRED or CBQ to implement in terms of its state requirements and algorithmic complexity.

2 ACTIVE QUEUE MANAGEMENT AND PACKET SCHEDULING

The default "best-effort" packet-forwarding service of IP is typically implemented in routers by a single, fixed-size, FIFO queue shared by all packets to be transmitted over an outbound link. The queue simply provides some capacity for tolerating variability in the load (*i.e.,* bursty traffic) on the outbound link. A short burst of packet arrivals may exceed the available bandwidth of the link even when the average load is well below the link bandwidth. However, when the load exceeds the available capacity of the link for sustained periods of time, the queue capacity is exceeded. Router implementations using a simple fixed-size FIFO queue typically just drop any packet that arrives to be enqueued to an already-full outbound queue. This behavior is often called *drop-tail* packet discarding. Braden *et al.* describe two important problems with the drop-tail behavior [2]. First, in some situations, many of the flows can be "locked-out," a condition in which a small subset of the flows sharing the outbound link can monopolize the queue during periods of congestion. Flows generating packets at a high rate can fill up the queue such that packets from flows generating packets at substantially lower rates have a higher probability of arriving at the queue when it is full and being discarded.

The second problem also occurs when the queue remains full or nearly full for sustained periods of time. When the queue is constantly full, latency is increased for all flows. Simply making the queue shorter will decrease the latency but negates the possibility of accommodating brief bursts of traffic without dropping packets unnecessarily. Two queue management policies, *random drop on full* [10] and *drop front on full* [12], address the lock-out phenomenon by causing packet drops to be spread over more flows, especially those that tend to dominate the queue content. These policies, however, still allow queues to remain full for sustained periods of time.

The latency problems associated with full queues can be addressed for responsive flows by dropping some packets before the queue fills. We use the term responsive flow to indicate any flow in which some end-to-end mechanism is used to detect packet loss and to adjust (reduce) the rate at which packets are sent in response to the loss. The classic example is, of course, the TCP congestion control mechanism [11] that is the essential mechanism that allowed the Internet to scale to today's reach while avoiding collapse from unconstrained congestion. Since responsive flows decrease the load they generate in response to drops, the queue should eventually cease to grow (depending on a variety of factors such as the round-trip latency for the individual flows). These types of pro-active approaches (random drop on full, drop front on full, and dropping prior to queue overflow) are referred to as *active queue management*.

2.1 Random Early Detection (RED)

RED is an active queue management policy that is intended to address some of the shortcomings of standard drop-tail FIFO queue management [5]. It addresses both the "lock-out" problem (by using a random factor in selecting which packets to drop) and the "full queue" problem (by dropping packets early, before the queue fills) for responsive flows.

The RED algorithm uses a weighted average of the total queue length to determine when to drop packets. When a packet arrives at the queue, if the weighted average queue length is less than a minimum threshold value, no drop action will be taken and the packet will simply be enqueued. If the average is greater than a minimum threshold value but less than a maximum threshold, an *early drop* test will be performed as described below. An average queue length in the range between the thresholds indicates some congestion has begun and flows should be notified via packet drops. If the average is greater than the maximum threshold value, a *forced drop* operation will occur. An average queue length in this range indicates persistent congestion and packets must be dropped to avoid a persistently full queue. Note that by using a weighted average,

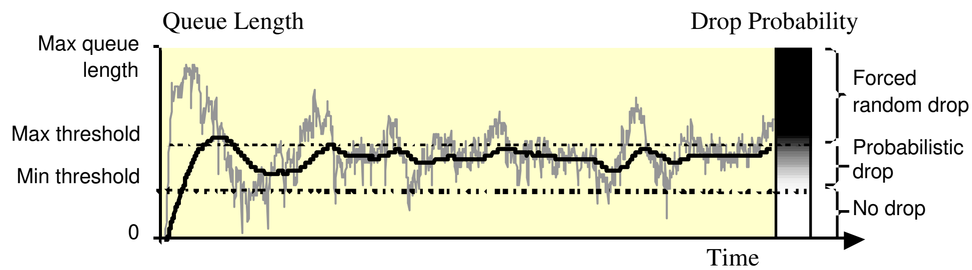


Figure 1: RED thresholds. Gray line indicates instantaneous queue length, black line indicates the weighted average queue length.

RED avoids over-reaction to bursts and instead reacts to longer-term trends. Furthermore, note that because the thresholds are compared to the weighted average (with a typical weighting of 1/512 for the most recent queue length), it is possible that no forced drops will take place even when the instantaneous queue length is quite large. For example, Figure 1 illustrates the queue length dynamics in a RED router used in our experiments. For the experiment illustrated in Figure 1, forced drops would occur only in the one short interval near the beginning when the weighted average reaches the maximum threshold. The *forced drop* is also used in the special case where the queue is full but the average queue length is still below the maximum threshold.

The *early drop* action in the RED algorithm probabilistically drops the incoming packet when the weighted average queue length is between the minimum and maximum thresholds. The probability that the packet will be dropped is relative to the current average queue length. In contrast, the *forced drop* action in the RED algorithm is guaranteed to drop a packet. However, the dropped packet is randomly selected from among all of the packets in the queue (including the one that arrived). During the drop phases of the RED algorithm, high bandwidth flows will have a higher number of packets dropped since their packets arrive at a higher rate than lower bandwidth flows (and thus are more likely to either be dropped during an early drop or have packets in the queue selected during the forced random drop phases). These mechanisms result in all flows experiencing the same loss rate under RED. By using probabilistic drops, RED maintains a shorter average queue length, avoiding lockout and repeatedly penalizing the same flow when a burst of packets arrives.

2.2 Misbehaving flows can dominate TCP traffic

An implicit assumption behind the design of RED is that all flows respond to loss as an indicator of congestion. When unresponsive flows consume a significant fraction of the capacity of the outbound link from the router, then RED can fail. RED fails in the sense that TCP flows can be locked out from the queue and experience high latency. In the worst case, unresponsive, high-bandwidth flows will continue to transmit packets at the same (or even at a higher) rate despite the increased drop rate due to RED. These high bandwidth, unresponsive flows will suffer more drops than lower bandwidth flows (including responsive flows that have reduced their load). However if these flows, either alone or in combination, consume a significant fraction of the capacity of the outbound link from the router, they will force TCP connections to transmit at minimal rates. Responsive flows, experiencing a high packet drop rate because of the high queue occupancy maintained by the unresponsive flows, will further reduce their traffic load. Figure 2 shows the result of an experiment designed to illustrate this effect on a 10 Mbps link. Figure 2 shows TCP's utilization of the outbound link from a router employing RED. The aggregate throughput of all TCP connections collapses when a single high-bandwidth UDP flow is introduced between time 60 and 110 (the experimental environment in which these data were measured is described in Section 3.2).

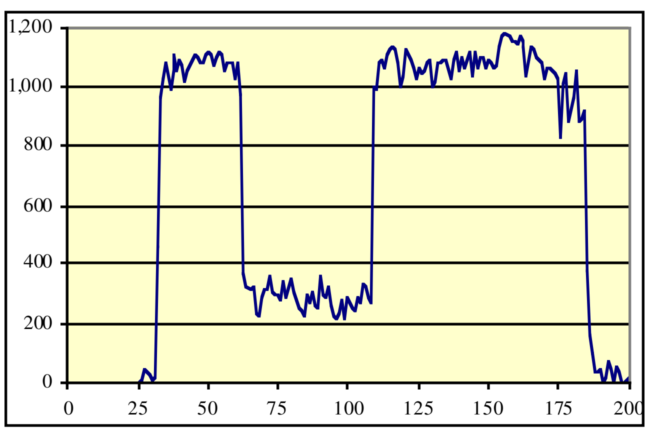


Figure 2: Aggregate TCP throughput with RED in the presence of an unresponsive, high-bandwidth UDP flow. (TCP throughput in kilobytes/second versus elapsed time in seconds.)

2.3 FRED – a proposal for fairness in buffer allocation

RED is vulnerable to unresponsive flows dominating a router's queue. Lin and Morris recognize this shortcoming of RED and proposed a scheme, called *Flow Random Early Detection* (FRED), to promote fair buffer allocation between flows [13]. To motivate FRED reconsider RED's response to congestion. Under RED, although higher-bandwidth flows incur a larger *number* of packet drops, on average, all flows experience the same loss *rate*. Flows experience the same loss rate because for a given average queue length, packets from all flows have the same drop probability. Therefore, two constant bit-rate flows that were generating loads of 10 Mbps and 1 Kbps on a 10 Mbps link during a period of congestion, may, for example, both see (on average) 10% of their packets dropped, leaving the flows with 9Mbps and 0.9Kbps of throughput, respectively. However, one could argue that the higher bandwidth flow is more responsible for the congestion and the 1 Mbps flow should be left untouched while the 10 Mbps flow is penalized.

FRED attempts to provide fair buffer allocation between flows, isolating each flow from the effects of misbehaving or non-responsive flows. FRED's approach is to impose uniformity during times of congestion by constraining all flows to occupying loosely equal shares of the queue's capacity (and hence receiving loosely equal shares of the outbound link's capacity). Moreover, flows that repeatedly exceed an average fair share of the queue's capacity are tightly constrained to consume no more than their fair share. This uniformity comes at a cost, however. Statistics must be maintained for every flow that currently has packets in the outbound queue of the router. These so-called "active flows" are allocated an equal share of the queue, which is determined by dividing the current queue size by the number of active flows. The number of packets a flow has enqueued is compared to the product of the flow's share value and a constant multiplier. This multiplier allows for non-uniform (bursty) arrival patterns among flows. A flow that exceeds the threshold including the multiplier is considered unresponsive and is constrained to its share (without the multiplier) until it has no more packets in the queue.

The results of this approach can be seen in the TCP Throughput graph in Figure 3. The traffic load is the same as that in the earlier experimental evaluation of RED. In particular, UDP blast is active from time 50 to time 100. While there is some decrease in TCP throughput, the overall performance is much better than that seen when simply using RED (Figure 1). In particular there is no congestive collapse. The difference in the results illustrated in Figures 1 and 2 is that in the FRED case, the unresponsive UDP flow is constrained to consume a fair share of the router's outbound queue. With hundreds of TCP connections (as part of this experimental set-up), we can estimate that there are a large number active flows (relative to the queue size of 60) at any given time, resulting in queue share on the order of 1-3 packets. Because the UDP flow is unresponsive (and high-bandwidth), it exceeds this share and is constrained to never occupying more than 1-3 slots in the queue. This results is a significantly higher level of packet loss for the unresponsive UDP flow than under RED (and hence higher throughput for all other well-behaved flows). Under RED, the unresponsive UDP flow could monopolize the queue and achieve significantly higher throughput. Under FRED, each active TCP flow gets the same number of buffer slots in the router queue as the unresponsive UDP flow does.

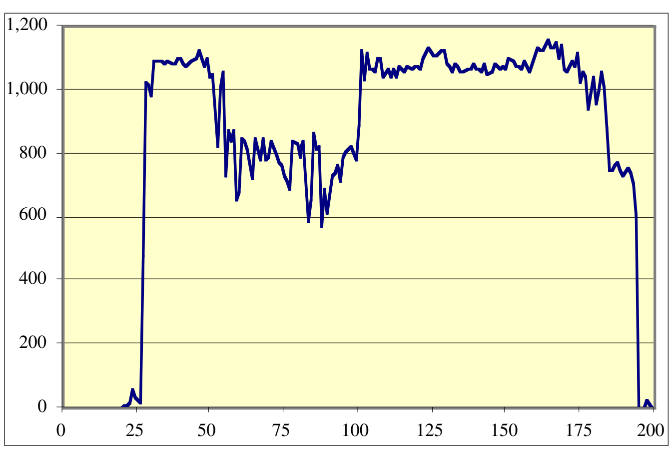


Figure 3: Aggregate TCP throughput under FRED in the presence of an unresponsive, high-bandwidth UDP flow. (TCP throughput in kilobytes/second versus elapsed time in seconds.)

FRED is the active queue management approach CBT most obviously resembles. The major difference is in the goals and the complexity of the two algorithms. FRED attempts to ensure fair allocation among all flows. CBT attempts to provide protection for tagged (multimedia) and TCP flows. Although fairness provides a form of protection, if the goal is strictly protection, a scheme like CBT is superior to FRED in terms of the performance of multimedia flows. FRED's major weakness, however, is the overhead associated with tracking active flows and keeping statistics (packet counts) for each active flow.

2.4 The consequences for multimedia applications

Most multimedia applications choose UDP as their underlying transport mechanism because they are concerned with throughput and latency rather than reliable delivery. Reliable delivery is the primary motivation for acknowledgements from receiver to sender. Without these acknowledgements (or some form of explicit feedback) packet loss cannot serve as an indication of congestion. As a result, a UDP flow (without some application-level adaptation mechanism) is necessarily "unresponsive" because, without a mechanism to detect packet drops, it has no indications to which it can respond. We are, therefore, interested in finding a queue management solution that works for TCP as well as for UDP flows that may or may not have application-level adaptation mechanisms. In this paper, we focus on one subset of UDP flows, interactive multimedia. We begin by noting that these flows, while typically low bandwidth themselves, are also vulnerable to the impact of high-bandwidth unresponsive flows (as demonstrated below). These flows are particularly sensitive to increases in the drop rate. Therefore, there is a requirement to not only isolate TCP streams from UDP, but also to isolate some types of UDP streams from one another. We are seeking an active queue management scheme that will maintain the positive features of RED, limit the impact of unresponsive flows, but still allow UDP flows access to a configurable share of the link bandwidth. Moreover, we seek to do this without having to maintain per flow state in the router. Our scheme, CBT (*Class Based Thresholds*), builds upon the drop thresholds of RED and the buffer allocations of FRED to provide a queue management policy that efficiently meets these goals.

3 CLASS-BASED THRESHOLDS

Our approach is to isolate TCP flows from the effects of all other flows by constraining the average number of non-TCP packets that may reside simultaneously in the queue. We also wanted to isolate classes of non-TCP traffic from one another, specifically isolating continuous media traffic from all other traffic. To do this we tag continuous media streams before they reach the router so that they can be classified appropriately. These flows are either self-identified at the end-system or identified by network administrators. (Nichols suggests one such tagging scheme using the *type-of-service* field in the IP header as part of an architecture for differentiated services. [14]) Statistics are maintained for these classes of traffic and their throughput is constrained during times of congestion by limiting the average number of packets they can have enqueued (thus limiting the fraction of router resources and link bandwidth they can consume). Untagged packets are likewise constrained by a (different) threshold on the average number of untagged packets enqueued. These thresholds only determine the ratios between the classes when all classes are operating at capacity (and maintaining a full queue). When one class is operating below capacity other classes can borrow that classes unused bandwidth.

Whenever a packet arrives, it is classified as being either *TCP*, *tagged* (continuous media), or *untagged* (all others). (For simplicity we assume only non-TCP packets are tagged.) TCP traffic is always subject to the RED algorithm as described above. For the other classes, the average number of packets enqueued for that class is updated and compared against the class threshold. If the average exceeds the threshold, the incoming packet is dropped. If the average does not exceed the threshold then the packet is subjected to the RED discard algorithm. Thus a non-TCP packet can be dropped either because there are too many packets from its class already in the queue or because the RED algorithm indicates that this packet should be dropped. TCP packets are only dropped as a result of performing a RED discard test. In all cases, although packets are classified, there is still only one queue per outbound link in the router and all packets are enqueued and dequeued in a FIFO manner.

We refer to the main CBT algorithm described above as "CBT with RED for all flows." We also consider the effect of not subjecting the non-TCP flows to a RED discard test. The motivation here comes from the observation that since the non-TCP flows are assumed (in the worst case) to be unresponsive, performing a RED discard test after policing the flow to occupy no more than its maximum allocation of queue slots provides no benefit to the flow and can penalize a flow even when that class is conformant. While the additional RED test may benefit TCP flows, they are already protected since the non-TCP flows are policed. Thus in this variant, called "CBT with RED only for TCP," only TCP flows are subjected to a

RED discard test. In addition, the weighted average queue length used in the RED algorithm is computed by counting only the number of TCP packets currently in the queue. This prevents an increase in TCP traffic from increasing the drop rate for a conformant tagged flow and it prevents the presence of the tagged and untagged packets from driving up the RED average, resulting in additional drops of TCP packets. For example, with thresholds of 10 packets for tagged and 2 packets for untagged traffic, these packets alone can maintain an average queue size of 12. Since these flows do not respond to drops, the resulting probabilistic drops only effect TCP traffic, forcing TCP to constantly respond to congestion. Removing these packets from the average reduces their impact on the TCP traffic. Thus when high-bandwidth UDP flows are active our first variant reduces to reserving TCP flows a minimum number of queue slots and performing the RED algorithm only on TCP. In this we way, we increase the isolation of the classes.

4 EXPERIMENTAL COMPARISON

We have implemented CBT and FRED within the FreeBSD kernel with ALTQ extensions [4]. ALTQ is a set of extensions to the default IP-layer packet queueing policies in a FreeBSD router to support development of experimental protocols, packet schedulers, and active queue management schemes. In addition to our active queue management implementations, we also used the ALTQ implementation of RED.

To test the implementation we have constructed a simple network consisting of two switched 100 Mbps Ethernet LANs that are interconnected by a 10 Mbps Ethernet. FreeBSD routers route traffic between the 100 Mbps Ethernets across a full-duplex 10 Mbps Ethernet as shown in Figure 4. The speed mismatch between the “edge” and “backbone” networks exists to ensure the backbone network is congested. A series of machines at the edges of the network establish a number of connections to machines on the opposite side of the network. Connections include a mix of TCP connections and UDP connections. In particular, several of the UDP connections are unresponsive multimedia flows generated by a series of Intel ProShare videoconferencing systems. Each ProShare connection generates approximately 210 kbps of traffic. For the TCP flows, the kernel on each machine introduces a delay in transmitting packets from each connection to simulate a larger round-trip time. This is done to avoid synchronization effects between the TCP flows and to create different bandwidth-delay products, encouraging connections to operate with different congestion window sizes.

Traffic is generated in all of the following experiments using a scripted sequence of flow initiations. This ensures comparable traffic patterns are generated in each experiment and hence the results of experiments may be compared. Initially 6 ProShare flows begin. Twenty seconds later, 240 TCP bulk transfers begin (all in the same direction). 30 seconds later, a single high bandwidth, unresponsive, UDP flow starts (in the same direction as the TCP flows). The UDP flow sends 1,000 byte packets at the maximum available rate (10Mbps) for approximately forty-five seconds. The bulk transfers finish forty-five seconds later. The ProShare applications terminate twenty seconds after the bulk transfers.

In separate experiments we ran this traffic pattern through a routers using RED, FRED, and CBT. In all of the experiments, the outbound queues in the routers have storage capacity for 60 packets (the FreeBSD default). The RED implementation was run with a minimum threshold of 15 queue elements (25% of the queue’s capacity), and a maximum threshold of 30 elements (50% of the queue’s capacity). (Thus when the weighted average queue length is less than 15 packets, arriving packets are queued. When the average length is between 15 and 30 packets, arriving packets are probabilistically dropped. When the average length is greater than 30 packets, a randomly selected packet is dropped when a new packet arrives.) These threshold values were selected based on recommendations of the developers of RED [5].

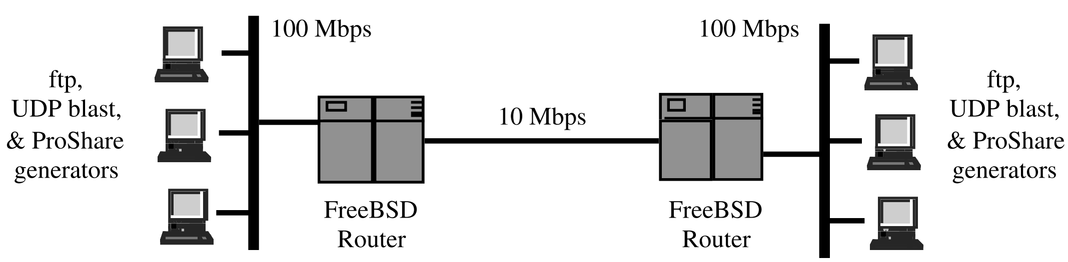
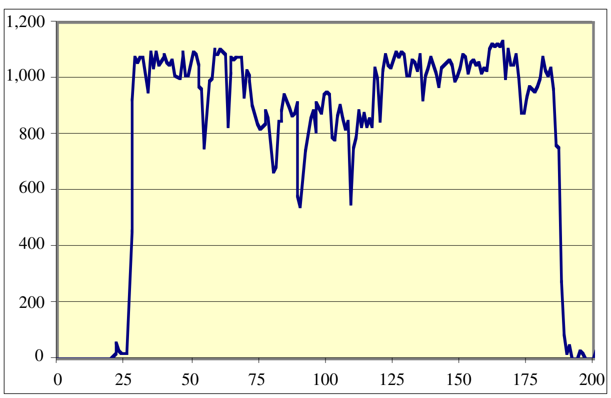


Figure 4: Experimental Network Setup

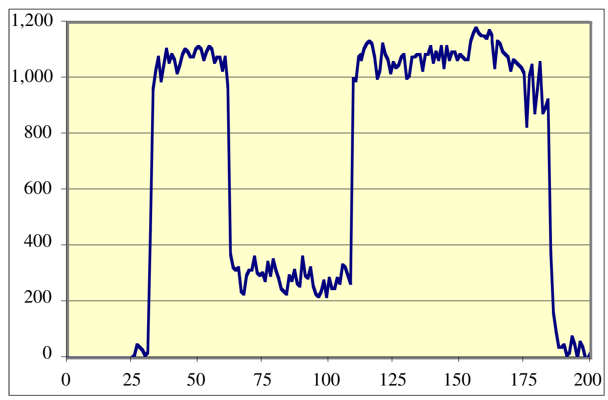
We also conducted the same set of experiments using a packet scheduling discipline, class based queueing (CBQ). Our CBQ parameters allocate 79% of the bandwidth to TCP and 21% to UDP. The UDP class is further divided into tagged and untagged subclasses receiving 16% and 5%, respectively, of the total bandwidth. All classes can borrow from their parent class. The 16% value for tagged traffic is based on the load of 1.6Mbps generated by the multimedia applications relative to the 10Mbps capacity of the bottleneck link. Untagged UDP is allocated a small fraction of the bandwidth (5%) and TCP is allocated the remainder.

The CBT implementation used a threshold for the untagged packet class of 2 packets, and a threshold for the tagged (multimedia) packet class of 10 packets. The threshold for the tagged class was determined based on the expected arrival rate of tagged packets relative to the expected drain rate of the queue. Given a drain rate, the threshold for the tagged class should ensure minimal packet loss (assuming applications generating tagged packets do not misbehave). Given our RED thresholds we expect the RED feedback mechanism to maintain a queue average of 26 packets. With a 10Mb link the drain rate is 800ns/packet. So a packet would take 22ms to transit a queue of length. From our knowledge of the multimedia traffic, we can expect an arrival rate of 210 pkts/second. From this we can determine that 10 packets should arrive during the 22ms it takes to transit the queue. As a result our threshold for tagged traffic is 10. By setting the tagged threshold to 10 we are "reserving" sufficient bandwidth for the ProShare flows.

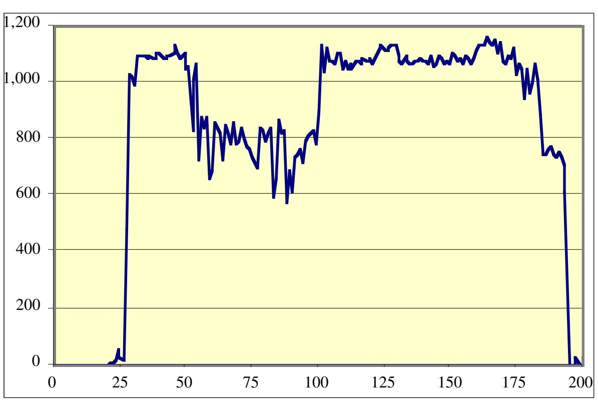
The choice of a threshold for the untagged ("all others") class is arbitrary but should be tied to the fraction of the router's capacity one is willing to let this class consume. Our selection of 2 as a threshold for untagged traffic essentially constrains untagged traffic to ~2/26 of the bandwidth during periods of congestion. Note that if there is no tagged or untagged traffic, TCP can still use 100% of the link capacity. Although we are "reserving" capacity for non-TCP flows, if



a) CBT with RED for all.



b) RED.



c) FRED.

Figure 5: Aggregate TCP throughput under "CBT with RED for all," plain RED, and FRED.
(TCP throughput in kilobytes/second versus elapsed time in seconds.)

the flows are not present, TCP flows may fill the queue (modulo only RED's policing). The reverse is true for tagged or untagged traffic. If all of the traffic is of one class that class can fully utilize the link because all of the packets in the queue (though shortened to 10 and 2 for tagged and untagged, respectively in this example) belong to that class.

In each experiment we collected throughput, loss, and latency data for all six of the ProShare streams. We also collected throughput data for all flows using *tcpdump* on the 10 Mbps bottleneck network.

4.1 CBT with RED for all

Our first approach is simply to add CBT to RED. All packets, including tagged and untagged, count in the averages used for RED and all packets are subject to the RED drop policies. Figure 5*a* shows the throughput of TCP with this scheme. As high bandwidth UDP traffic was introduced, TCP throughput is degraded. The degradation is far from the collapse under RED (Figure 5*b*), and comparable than that under FRED (Figure 5*c*). The loss rate for multimedia under CBT is 23.8% (Table 1). This is an improvement of 7 percentile over RED and 12 percentile better than FRED's loss rate but it is still unsatisfactory. However, the primary factor for the high drop rate in tagged traffic is the RED drop test, not the CBT thresholds. We illustrate this below with *CBT and RED for TCP only*. But, *CBT with RED for All* does a better job of isolating TCP and continuous media (tagged packets) from high-bandwidth unresponsive flows than RED. Further its performance is also comparable to FRED's and CBT provides a lower drop rate.

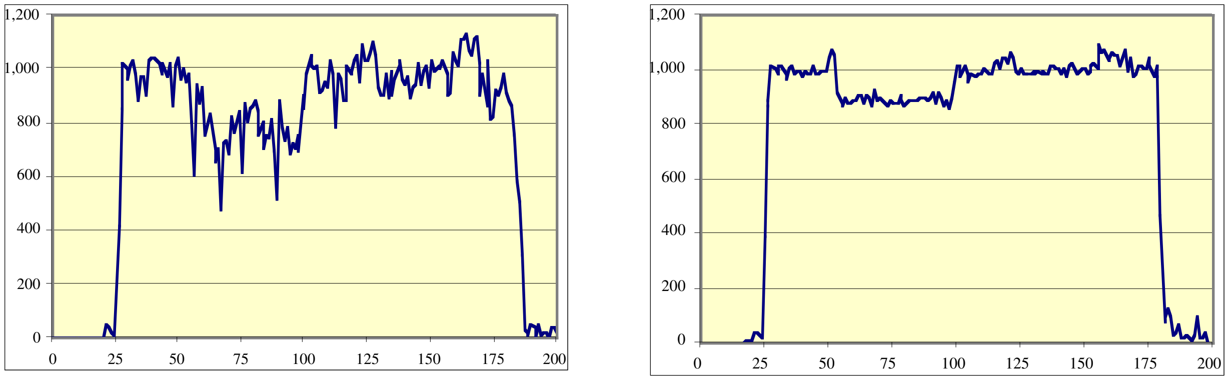
4.2 CBT with RED only for TCP

In CBT with RED when the threshold sizes are too large, particularly for the untagged class, those classes represent a large share of the average queue size. As a result, non-TCP traffic can dominate the queue while TCP is forced to reduce its load to essentially zero. While these parameters can be tuned to avoid this situation, the range is limited. Also, the tagged and untagged classes are in double jeopardy, subject to the thresholds for their class as well as the RED drop policies. Tagged (or untagged) traffic that is well within its threshold may be dropped because of the RED drops instigated by TCP. As a result, we need to isolate the tagged and untagged classes from TCP. To do this, we apply RED only to TCP flows. In the next experiment only TCP packets are subject to the RED threshold tests and only TCP packets count in the RED queue length average. The tagged and untagged classes are constrained only by their thresholds. In effect, this results in a true allocation of bandwidth for the tagged and untagged classes where there was only a probabilistic allocation before. The RED test is no longer applied to the tagged traffic, only the threshold test for this class. The expectation is that the number of drops for non-TCP packets will decrease while TCP performance improves. While the TCP throughput remained essentially the same (Figure 6*a*), the drop rate for multimedia improved substantially, from 23.8% to 1.3% (see Table 1). The RED test is no longer applied to the tagged traffic, only the threshold test for this class. Since the tagged traffic does not exceed its threshold it is rarely dropped.

One issue with removing non-TCP traffic from the RED average is how to adjust the queue size and RED thresholds. One alternative is to leave the queue size unchanged but reduce the RED queue limit to the queue size minus the slots allocated to tagged and untagged traffic. For example, with a queue of 60, a marked threshold of 10 and an unmarked threshold of 2, the effective RED queue limit reduces to 48 with minimum and maximum thresholds of 12 and 24 respectively. This approach should result in roughly the same average queue length but possibly at the cost of some

Table 1: Average per flow packet loss rate for tagged (ProShare) packets.

Queue Management Scheme	Drop Rate for Continuous Media
RED	30.0%
FRED	35.7%
CBT with RED for all	23.8%
CBT with RED for TCP only	1.3%
CBQ	0.0%



a) CBT with RED for TCP only. *b*) CBQ.

Figure 6: Aggregate TCP throughput under "CBT with RED for TCP only" and CBQ (TCP throughput in kilobytes/second versus elapsed time in seconds.)

capacity to accommodate bursty TCP traffic. The other alternative would be to increase the queue size by the number of slots allocated to tagged and untagged traffic. For the example above, that results in a queue of size 72, and RED thresholds unchanged. The result here is a longer average queue, resulting in increased latency. We conducted experiments with both alternatives. The results shown are for experiments where we adjusted the RED thresholds. The latency and drop rate in the other case (not shown) are marginally higher, as expected, with the TCP throughput unchanged.

CBQ, with its scheduling mechanisms, has a 0.0% drop rate since the multimedia traffic is staying within its 16% traffic allocation during times of congestion. Additionally, the throughput for TCP (Figure 6b) is higher (approximately 79% of the link capacity during periods of congestion.) It is important to note that CBT is able to approximate this performance.

4.3 Other performance metrics — Latency

While throughput/loss is a key performance metric, latency for interactive multimedia applications is often more important. As noted earlier, minimizing latency is one of the primary motivations for avoiding full queues. Figure 7 shows the end-to-end latency observed on a representative ProShare stream for CBQ, FRED, and CBT using RED for all packets. The other CBT results are comparable. The CBQ latency is less than 5 ms because packets in the tagged class are being serviced essentially as fast as they arrive since they are allocated 1.6Mbps and arriving at the same rate. The only delay is the packet transmission time. FRED, on the other hand, has an average latency of approximately 30ms while CBT averages around 25ms of latency. The primary reason for this is the shorter average queue length under CBT.

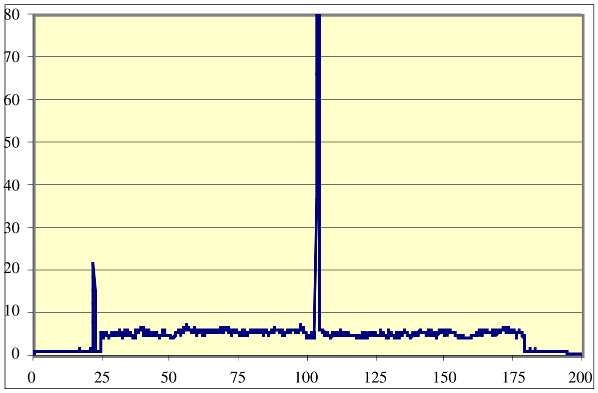
5 COMPARISON TO OTHER SCHEMES

5.1 Class Based Queueing

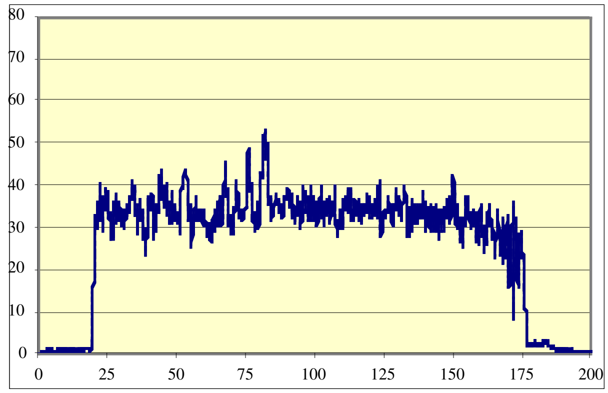
A more general approach that achieves many of the objectives considered here is found in models for link sharing such as *Class-Based Queuing* (CBQ) [7]. CBQ achieves isolation for classes of flows by limiting each class to a configurable share of link bandwidth. Implementations of CBQ define a separate queue for each of the traffic classes sharing a link. The general framework proposed for link sharing in [7] also includes provisions for different types of packet scheduling mechanisms (notably priority scheduling for real-time multimedia classes) and for using RED (or a similar mechanism) for regulating the effects of congestion in a best-effort traffic-class queue. CBT is a less general mechanism but achieves a good approximation to isolation and protection for a limited number of flow classes while preserving most of the simplicity of a single FIFO queue.

5.2 Other buffer management schemes

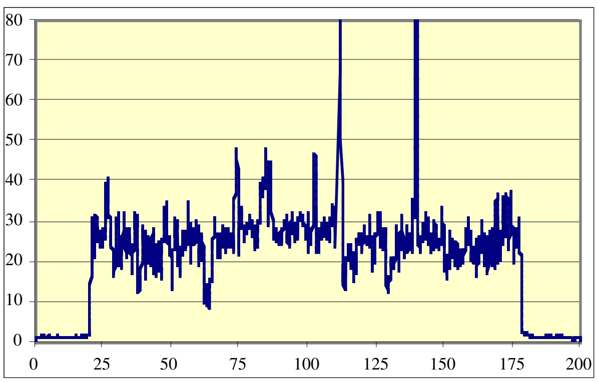
Guerin proposes a flow isolation and bandwidth sharing mechanism based on assigning a queue occupancy threshold in a single FIFO queue to individual flows (or flow aggregations)[9]. The amount reserved is sufficient to provide throughput



a) CBQ.



b) FRED.



c) CBT with RED for all.

Figure 7: ProShare latency under CBQ, FRED, and CBT.
(ProShare latency in milliseconds versus elapsed time in seconds.)

rate guarantees along with lossless service for flows that are conformant with their peak-rate flowspec. Their results show this can be achieved at considerable cost in added queue capacity (up to 6 times more) necessary to achieve the same link utilization obtained with an unpartitioned FIFO queue. CBT makes no attempt achieve throughput rate guarantees or lossless service and, as shown above, operates effectively without requiring increased queue capacity.

5.3 Complexity Comparison

Complexity, both computational and state, is a key issue in routers. Given the high packet-rate that routers must maintain, per-packet overhead must be minimized. Further, any additional state requires memory that could have been used to accommodate larger traffic bursts. Overhead for each packet is a significant factor in performance. The experiments we have shown did not highlight point because the routers, 200MHz Pentium Pro machines, have more than enough processing power to manage the traffic for the 10 Mbps Ethernet bottleneck. In this section we compare the complexity of FRED, CBQ, and CBT.

To compare the relative complexity of the three algorithms, we will briefly walk through the overhead associated with each. Both algorithms require examining the header of the packet in order to determine how to classify the packet. In the case of CBT and CBQ this classification, into one of three classes: *TCP*, *tagged*, or *untagged*, is an operation that takes constant time. In the case of FRED the packet must be classified by which flow it is associated with. This classification is conceptually O(N) where N is the number of active flows. Optimizations such as hashing or indexing can improve the computational complexity and run-time execution time. However, such improvements come at the cost of additional memory. Both FRED and CBT require updating statistics on every enqueue and dequeue operation. However, in the case of CBT the number of statistics involved is constant, one set for each of the three classes. For FRED there are packet counts, strike counts, and a 5-tuple to identify the associated flow for every active flow (O(n)). Finally, the test to make the choice whether or not to drop a packet is simpler in CBT. The packet average for that class is simply compared to the threshold, and then only if the packet is not in the TCP class. However, in FRED's case, there are a few mathematical operations to determine the current acceptable thresholds based on current average queue size and the current number of active flows. There are also several tests to determine the current state of the associated flow (relaxed or tightly constrained) in FRED. This test is applied to all packets. The dequeue operation for CBQ is the most expensive because of the overhead of selecting the next packet from one of several queues to be transmitted based on the weights of each, the state of the other classes (for borrowing), and the current service levels each class is maintaining. The CBT algorithm is less complex and requires less state than FRED or CBQ.

6 CONCLUSIONS

Ultimately, traffic with strong quality of service requirements will be best served by packet scheduling techniques such as CBQ. However, packet scheduling has yet to be widely embraced or deployed within the Internet. In contrast active queue management is the widely accepted mechanism for managing queues in routers servicing best effort traffic. Implementations such as RED and FRED have demonstrated better performance than traditional drop tail mechanisms. In this paper we have introduced a new active queue management scheme, CBT that seeks to provide the congestion avoidance benefits of RED while providing protection for TCP and a better-than-best-effort service for well-behaved UDP flows. We have explained how we aggressively constrain non-TCP traffic to limit its impact on TCP traffic and other classes of non-TCP traffic, such as continuous media. We have empirically compared CBT to RED and FRED. These experiments demonstrated that:

- TCP receives a better share of bandwidth when facing misbehaving or unresponsive multimedia flows with CBT than with RED. Performance with CBT is comparable with FRED.
- Using CBT, the number of drops on low bandwidth tagged flows is substantially lower than when RED or FRED are used.

Further, we conducted the same experiments using a packet scheduling mechanism, CBQ, to provide a standard to measure these schemes against. We have shown that using a combination of thresholds on queue occupancy and RED we can approach the class isolation and multimedia drop rate of CBQ.

Moreover, these results are achieved with less complexity than FRED or CBQ. FRED maintains state for every flow. CBQ must schedule ever packet dequeue. CBT also offers flexibility in assigning router resources to different classes of traffic instead of the uniform distribution offered by FRED. CBT shows promise as an extension to conventional RED to constrain unresponsive traffic and to support self-identified multimedia flows in today's Internet.

7 ACKNOWLEDGEMENTS

The authors wish to thank the anonymous referees for their suggestions that enhanced the quality of this paper.

8 REFERENCES

1. R. Braden, Ed., Requirements for Internet Hosts-Communication Layers, RFC-1122, October 1989.
2. B. 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, & L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, Internet draft, work in progress, 1998.
3. S. Cen, C. Pu, J. Walpole, *Flow and Congestion Control for Internet Streaming Applications*, Proc. SPIE/ACM Multimedia Computing and Networking '98, San Jose, CA, January 1998, pages 250-264.
4. K. Cho, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers*, Accepted to USENIX '98, Annual Technical Conference, New Orleans, LA, June 1998.
5. S. Floyd, & V. Jacobson, *Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Trans. on Networking*, V.1 N.4, August 1993, p. 397-413.
6. S. Floyd, *TCP and Explicit Congestion Notification*, ACM Computer Communications Review, 24(5):10-23, Oct. 1994.

7. S. Floyd & V. Jacobson, *Link-Sharing and Resource Management Models for Packet Networks*, IEEE/ACM Transactions on Networking, V.1, N.4, August 1995, pp. 365-386.

8. S. Floyd, S., & K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, February 1998. (Under submission to *IEEE/ACM Trans. on Networking*.)

9. R. Guerin, S. Kamat, V. Peris, and R. Rajan*, Scalable QoS Provision Through Buffer Management*, Proceedings of SIGCOMM'98, (to appear).

10. E. Hashem, *Analysis of Random Drop for Gateway Congestion Control,* Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA, 1989, p. 103.

11. V. Jacobson, *Congestion Avoidance and Control*, ACM SIGCOMM '88, August 1988.

12. T.V. Lakshman, A. Neidhardt, T. Ott, *The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features,* Proc. Infocom 96, pp. 1242-1250.

13. D. Lin & R. Morris, *Dynamics of Random Early Detection*, Proc. SIGCOMM '97.

14. K. Nichols, V. Jacobson , & L. Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, Internet draft, work in progress, 1997.