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*.

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,

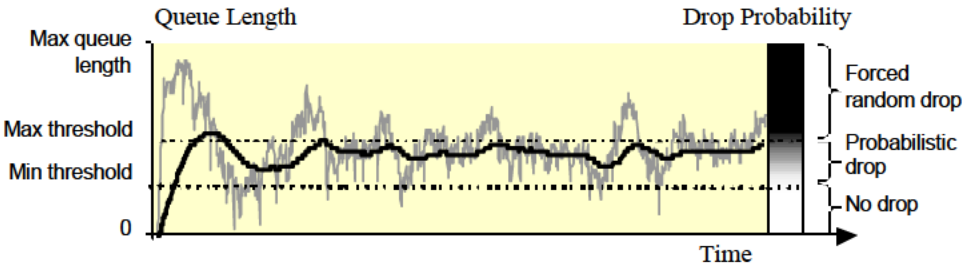

Figure 1: RED thresholds. Gray line indicates instantaneous queue length, black line indicates the weighted average queue length.

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).

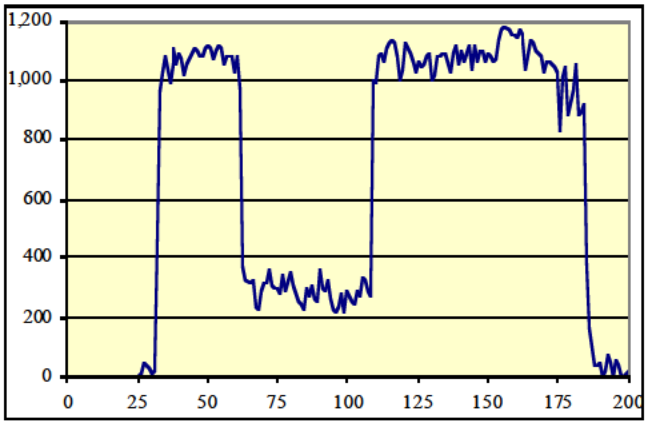

Figure 2: Aggregate TCP throughput with RED in the presence of an unresponsive, high-bandwidth UDP flow. (TCP throughput in kilobytes/second versus elapsed time in seconds.)

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.

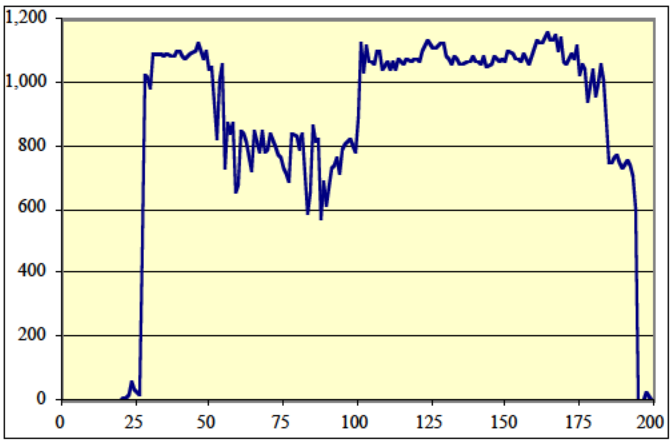

Figure 3: Aggregate TCP throughput under FRED in the presence of an unresponsive, high-bandwidth UDP flow. (TCP throughput in kilobytes/second versus elapsed time in seconds.)

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.