Controlling High-Bandwidth Flows at the Congested Router

Ratul Mahajan[†*] Sally Floyd[‡] David Wetherall[†]

† University of Washington
Seattle, WA

‡AT&T Center for Internet Research at ICSI (ACIRI)
Berkeley, CA

Abstract

FIFO queueing is simple but does not protect traffic from high-bandwidth flows, which include not only flows that fail to use end-to-end congestion control, but also short round-trip time TCP flows. At the other extreme, per-flow scheduling mechanisms provide max-min fairness but are more complex, keeping state for all flows going through the router. This paper presents RED-PD, a mechanism that combines simplicity and protection by keeping state for just the high-bandwidth flows. RED-PD uses the packet drop history at the router to detect high-bandwidth flows in times of congestion and preferentially drops packets from these flows. This paper discusses the design decisions underlying RED-PD. We show that it is effective at controlling high-bandwidth flows using a small amount of state and very simple fast-path operations.

1. Introduction

The dominant congestion-control paradigm in the Internet is FIFO (First In First Out) queueing at routers in combination with end-to-end congestion control. FIFO queueing is simple to implement and well-suited to the heterogeneity of the Internet. But it provides little protection from high-bandwidth flows that consume excessive bandwidth at the expense of other flows at the router. These high-bandwidth flows can be TCP flows with small round-trip times (a TCP flow's throughput is inversely proportional to its RTT), or worse, flows not using end-to-end congestion control. During congestion it is important to control the high-bandwidth flows to improve the performance of the rest of the traffic.

At the other extreme, per-flow scheduling mechanisms provide max-min fairness, but keep state for all the flows. This is an unnecessarily complex solution because most of the flows going through the router are "Web mice" (short HTTP flows). Moreover, the level of fairness provided by such schemes is not required for best-effort traffic, which is the focus of this paper.

In this paper, we present RED-PD, RED [9] with Preferential Dropping. We show that by keeping state for only the high-bandwidth flows and controlling their throughput during congestion, a router can combine the simplicity of FIFO with the protection of full max-min fair techniques. We call this approach *partial flow state*, and show its effectiveness in an environment like the Internet, dominated by end-to-end congestion control and a skewed bandwidth distribution among flows, in which a small fraction of flows account for a large fraction of bandwidth.

RED-PD identifies high-bandwidth flows using the RED packet drop history. The packet drops from active queue management are a reasonably unbiased sample of the incoming traffic [7], and at the same time represent flows that have been sent congestion indications by the router. Flows above a configured *target bandwidth* are identified and *monitored* by RED-PD.

RED-PD controls the throughput of the monitored flows by probabilistically dropping packets from them at a prefilter placed before the output queue. The dropping probability, computed using the identification mechanism itself, is such that the rate of the flow into the output queue is reduced to the target bandwidth. RED-PD suspends preferential dropping when there is insufficient demand from other traffic in the output queue, for example, when RED's average queue size is less than the minimum threshold.

Figure 1 illustrates RED-PD's impact on incoming traffic. Assume that flows are identified when their arrival rate is more than the target bandwidth T, and, when monitored, are restricted to T if there is enough demand from other flows. RED-PD has no effect when T is set higher than the maximum arrival rate of a flow. As T is pushed down, the bandwidth obtained by the monitored flows (A) will be curtailed. This reduces the *ambient* drop rate, defined as the drop rate at the output queue, and enables the non-monitored flows (B, C, D) to receive more bandwidth.

In the next section we discuss existing proposals that use preferential dropping to improve fairness among flows.

[*] Much of the work was done while the author was at ACIRI.

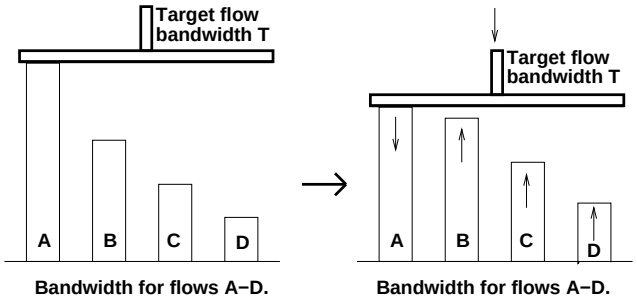

Figure 1. Restricting flows to a target bandwidth T.

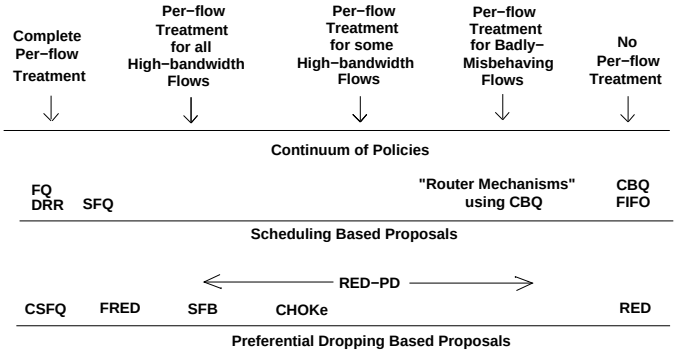

Figure 2. A continuum of per-flow treatment at the queue. FQ [4], DRR [21], SFQ [13], and Router Mechanisms [6] are scheduling-based schemes, whereas CSFQ [22], FRED [10], SFB [5], RED-PD, and CHOKe [18] use preferential dropping.

Section 3 discusses some trace-based results showing that controlling the small number of high-bandwidth flows can significantly improve the performance of other traffic. Section 4 describes RED-PD in detail, and Section 5 evaluates it using simulation. A discussion on complexity of RED-PD is contained in Section 6, and on the problem of unresponsive flows in Section 7.

2. Related Work

Mechanisms for per-flow treatment at the router can be classified as based on either scheduling or preferential dropping. *Scheduling* approaches place flows in different scheduling partitions, and the scheduling mechanism determines the bandwidth received by each partition. In contrast, *preferential dropping* mechanisms vary the dropping rate of a flow to control its throughput. The technical report [11] discusses the advantages of preferential dropping over scheduling.

Figure 2 classifies the existing approaches based on their control approach, and roughly places them along the continuum of per-flow treatment. The amount of flow state kept increases from right to left. In the remainder of this section we only discuss proposals that use preferential dropping, the approach taken by RED-PD.

This paper is in some sense a successor to [6], in which Floyd and Fall discuss mechanisms for identifying high-bandwidth flows from the RED drop history. However, their approach is limited by the choice of aggregate scheduling-based mechanisms (CBQ) instead of the per-flow preferential dropping mechanisms used by RED-PD.

RED-PD draws heavily from Core-Stateless Fair Queuing (CSFQ) [22] and Flow Random Early Detection (FRED) [10], two approaches that use per-flow preferential dropping in concert with FIFO scheduling. The goal of CSFQ is to achieve fair queuing without using per-flow state in the core of an *island* of routers (an ISP network, for instance). On entering the network, packets are marked with an estimate of their current sending rate. A core router estimates a flow's fair share and preferentially drops a packet from a flow based on the fair share and the rate estimate carried by the packet. Key differences from our approach are that CSFQ requires an extra field in the packet headers, and all the routers within the island need to be modified.

FRED maintains state at the router only for those flows that have packets currently in the queue. The dropping probability of a flow depends on the number of buffered packets from that flow. FRED's fair allocation of buffers yields different fairness properties from a fair allocation of bandwidth [22].

In CHOKe [18], an incoming packet is matched against a random packet in the queue. If they belong to the same flow, both packets are dropped, otherwise the incoming packet is admitted with a certain probability. The scheme tries to leverage the fact that high-bandwidth flows are likely to have more packets in the queue. CHOKe is not likely to perform well when the number of flows is large and even the high-bandwidth flows have only a few packets in the queue. The simulations in the paper show that it achieves limited performance; for example, the high-bandwidth UDP flows get much more than their fair share.

[17] presents an approach that is an outgrowth of CSFQ and CHOKe. The router keeps a sample of arriving traffic, and an incoming packet is matched against this sample. The dropping probability of the incoming packet is determined by the number of packets in the sample from the same flow.

Stochastic Fair Blue (SFB) [5] relies on multiple levels of hashing to identify high-bandwidth flows. As the authors state in their paper, the scheme works well when there are

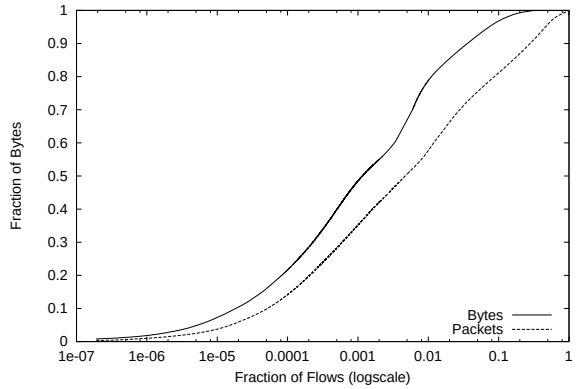

Figure 3. Skewedness of bandwidth distribution.

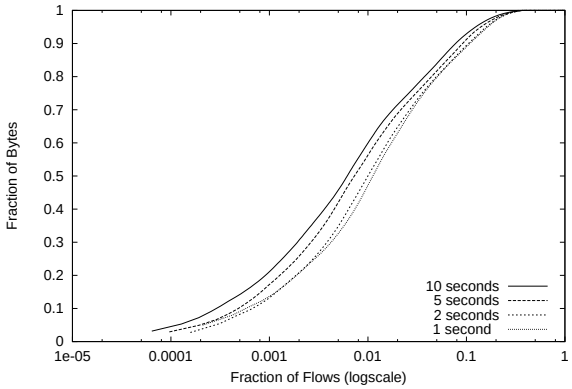

Figure 4. Skewedness over smaller time scales.

only a few high-bandwidth flows. In the presence of multiple high-bandwidth flows SFB ends up punishing even the low bandwidth flows as more and more bins get polluted.

The scheme presented in [2] drops packets based on the buffer occupancy of the flow, and ERUF [1] uses source quench to have undeliverable packets dropped at the edge routers. SRED [15] relies on a cache of recently seen flows to determine the high bandwidth flows.

What makes RED-PD different from other schemes is that it explicitly leverages the skewed bandwidth distribution in the Internet to improve the performance of low-bandwidth flows using a small amount of state, and has a predictable effect on the traffic going through the router.

3. Why does a Partial Flow State Approach Work?

In this section we present trace results showing that RED-PD's approach of keeping state for only high-bandwidth flows can be effective. The traces that we examined exhibit the same characteristics as found by others [3], namely, that a small fraction of flows are responsible for most of the bytes sent. We further show that identifying and preferentially dropping from this small number of flows is a powerful approach, since controlling the throughput of these flows results in a significant decrease in the ambient drop rate rate and leads to a higher throughput for other flows.

Figure 3 shows results from a one-hour-long trace taken from UCB DMZ in August 2000. It shows the fraction of flows responsible for a given fraction of bytes and packets in the trace. A flow here is defined by the tuple (source IP, source port, destination IP, destination port, protocol). A flow was timed out if it was silent for more than 64 seconds (the results with different timeout values are similar). It is clear from the graph that a mere 1% of the flows accounted for about 80% of the bytes and 64% of the packets.

Figure 4 plots the same information for shorter (than flow lifetimes) time windows. It shows the fraction of flows responsible for a given fraction of bytes and packets in a given time interval. We can see that the skewedness holds not only for long time periods but also for smaller time scales, at which one is likely to identify the high-bandwidth flows.

Another property necessary for an identification based approach to be effective is that the high-bandwidth flows in a given interval be a good predictor of the high-bandwidth flows in the next interval. Figure 5 plots the fraction of bandwidth consumed in the subsequent interval by flows that accounted for a particular amount of bandwidth in the current interval. For example, from the graph in Figure 4 we can see that in a 5-second interval, 1% of the flows sent close to 50% of the bytes. Figure 5 tells us that these flows were responsible for 36% of the bandwidth in the next 5-second interval.

4. RED-PD

There are two components in RED-PD: identifying high-bandwidth flows and controlling the bandwidth obtained by these flows. We describe each in turn.

4.1. Identifying High Bandwidth Flows

RED-PD uses the RED drop history (mark history in the presence of ECN [19]) to identify high-bandwidth flows. Since RED drops are probabilistic, and not the result of a buffer overflow, they can be considered as reasonably random samples of the incoming traffic [7]. Moreover, the drop

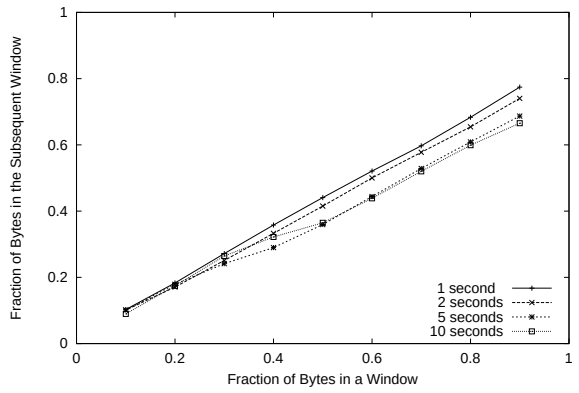

Figure 5. Predictive nature of bandwidth usage.

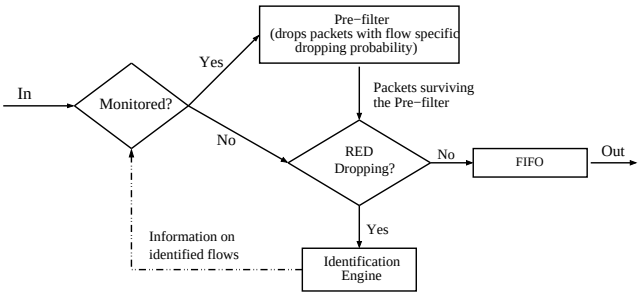

Figure 6. Architecture of a RED-PD router.

history represents flows that have been sent congestion signals. Thus, the drop history can be used to identify a high-bandwidth flow and to confirm that the identified flow has in fact received loss events.

The *target bandwidth*, above which a flow is identified, is defined as the bandwidth obtained by a *reference TCP* flow with the *target RTT* R and the current drop rate p at the output queue. Let $f(r,p)$ denote the average sending rate in pkts/s of a TCP flow with an RTT r and a steady-state packet drop rate p. From the deterministic model of TCP in [6] (reasons for choosing this equation instead of the more precise one given in [16] are discussed in the Appendix of [11]), we have:

$$f(r,p) \approx \frac{\sqrt{1.5}}{r\sqrt{p}}. \quad (1)$$

RED-PD's goal is to identify flows that are sending more than $f(R,p)$, the reference TCP flow's rate.

In the deterministic model with periodic packet drops, a TCP congestion epoch contains exactly one packet drop, and therefore contains $\frac{1}{p}$ packets. Hence, the *congestion epoch length* $CL(r,p)$, in seconds, of a TCP flow with RTT r with drop rate p is

$$CL(r,p) = \frac{1}{f(r,p)p} = \frac{r}{\sqrt{1.5p}} \quad (2)$$

Flows sending at a rate higher than $f(R,p)$ will have, on average, more than one drop in $CL(R,p)$ seconds, given a steady-state packet drop rate p. RED-PD maintains the packet drop history over $K \times CL(R,p)$ seconds, for some small integer K. The high-bandwidth flows will roughly have K or more drops in this history. RED-PD partitions the history into multiple lists containing drops from consecutive intervals of time. RED-PD keeps M lists, where the length of a list is

$$\frac{K}{M}CL(R,p) = \frac{K}{M}\frac{R}{\sqrt{1.5p}} \quad (3)$$

Instead of identifying flows with K or more losses in a history of $K \times CL(R,p)$ seconds, RED-PD identifies flows with losses in at least K of M lists. Because of the use of multiple lists, flows with losses concentrated in fewer lists are not identified[1]. There are several reasons why a flow might have multiple losses but not spread over K or more lists: because a single congestion event for that flow was composed of multiple drops from a window of data; because the flow reduced its sending rate after several RTTs with drops; or because a low-bandwidth flow got unlucky and suffered more than its share of drops.

In our simulations we use $K = 3$ and $M = 5$ (with $R = 40ms$ and $p = 1\%$, this corresponds to drop history of about 1 second). The technical report [11] discusses the guidelines for choosing these parameters, and also shows the advantages of using multiple lists over a single list.

4.2. Preferential Dropping

Figure 6 shows the architecture of a RED-PD router. Preferential dropping is done using a pre-filter in front of the output queue. Packets from the monitored flows are dropped in the pre-filter with a probability dependent on the excess sending rate of the flow. Unmonitored traffic is put in the output queue directly.

The light-weight mechanism shown above: a) not only protects unmonitored traffic from the monitored flows, but also provides relative fairness among the monitored flows; b) does not starve the monitored flows like "leftover bandwidth" approaches; c) does not protect the monitored flows

[1] Potentially, this opens up the possibility of evading the identification mechanism by sending big bursts such that drops are confined to fewer lists. But is unlikely as it would require a precise knowledge of the drop rate at the router. If this proved to be a problem, it could be addressed by extending the identification mechanism to flows with a large number of losses even if those losses are present in fewer lists.

```
Parameters
 max_decrease: max probability reduction
                in one step
 P_minThresh:  max probability to free
                a monitored flow

foreach f  (monitored flows that don't
            appear in any of the M drop lists)
   P = dropping probability of f
   if (P > 2*max_decrease)
      P -= max_decrease
   else
      P = P/2
   if (P ≥ P_minThresh)
      dropping probability of f = P
   else
      release f
```

Figure 7. Pseudocode for reducing a flow's dropping probability.

```
Variables
 avg_drop_count: average number of drops for
                 flows identified in this round
      p:         current ambient drop rate

foreach f  (flows that appear in at least K
            of M drop lists)
   if (f is monitored)
      P_f = dropping probability of f
   else
      P_f = 0
   drop_f = number of drops of f
   P_delta = (drop_f/avg_drop_count)*p
   if (P_delta > P_f + p)
      P_delta = P_f + p
   dropping probability of f += P_delta
```

Figure 8. Pseudocode for increasing a flow's dropping probability.

from the general congestion at the link, because the output queue does not differentiate between flows. Disguised protection can occur with approaches that reserve a fixed amount of bandwidth without regard to the level of unmonitored traffic.

There would be a danger of transient link underutilization if packets were dropped in the pre-filter despite low demand at the output queue. To avoid this, the pre-filter does not drop packets from monitored flows when there is insufficient demand at the output queue, as measured by RED's average queue size.

Instead of directly measuring the arrival rate of the monitored flows for computing the dropping probability, RED-PD bases the pre-filter dropping probability on the identification mechanism itself. The identification process only considers drops at the output queue, not in the pre-filter. Thus, the identification process is concerned with the flow's arrival rate to the output queue, not the arrival rate at the router itself; the two quantities would be different for a monitored flow.

A monitored flow will be identified again if its dropping probability is not high enough to reduce its rate to the output queue to less than $f(R,p)$. The dropping probability is increased for such flows. If the flow cuts down its sending rate and does not appear in any of the last M drop lists, its dropping probability is decreased. With this iterative increase and decrease, RED-PD settles around the right pre-filter dropping probability for a monitored flow.

Changes to the dropping probability are not made until a certain time period has elapsed after the last change to ensure that the flow has had time to react to the last change. Since we are not using a token bucket, we have to make iterative changes to compute the dropping probability required to reduce the flow's rate to $f(R,p)$, the target bandwidth. And because we are interacting with the flow's end-to-end congestion control, our iterations must be on the time scale of round-trip times.

The dropping probability for a monitored flow is not changed when the the flow appears in at least one but fewer than K of the M drop lists. This provides the necessary hysteresis for stabilizing the dropping probability. If a flows reduces its sending rate enough to make the dropping probability negligible, it is unmonitored altogether.

The pseudocode for reducing the dropping probability is given in Figure 7. The reduction in the dropping probability is bounded by a maximum allowable decrease in one step to reduce oscillations. These oscillations could result from the reactions of the flow's end-to-end congestion control mechanisms to packet drops, or from the imprecision of using a flow's packet drop history as an estimate of its arrival rate. That is, the absence of the flow in all the drop-lists could be the result of it getting lucky, rather than a reduction in its sending rate. In such cases *max_decrease* ensures that control over a monitored flow is not loosened by a large amount in one step.

Figure 8 shows the pseudocode for increasing a flow's dropping probability. The equation for a flow's increase quantum, P_{delta}, takes into account both the ambient packet drop rate and the relative sending rate of the monitored flow, as inferred from the ratio of drops. The increase quantum is large when the ambient drop rate is high, or the flow has a higher sending rate. To avoid abruptly increasing the drop rate experienced by a flow, the increase quantum is lim-

ited to the flow's existing drop rate (which doubles the drop rate).

5. Evaluation

In this section, we evaluate RED-PD's effectiveness in controlling high-bandwidth flows and protecting other traffic at the router. Since identification is the first step, in §5.1 we study RED-PD's effectiveness in identifying high-bandwidth flows. RED-PD's ability to enforce fairness using the iterative increase and decrease of a flow's dropping probability is investigated in §5.2. It is important that RED-PD react reasonably promptly to changes in a flow's sending rate, a property we analyze in §5.3. Finally, in §5.4, we demonstrate how the choice of the target RTT R effects the degree of fairness and amount of state kept by the router.

Additional simulations, not included due to space considerations, can be found in [11]. These include: (1) a simulation with Web traffic that shows RED-PD can reduce the average completion time of a Web request by controlling the throughput of long-lived high-bandwidth flows; (2) a simulation showing RED-PD does not negatively impact traffic belonging to a different congestion control model like TFRC [8]; (3) a simulation with multiple congested links; and (4) a simulation showing byte mode operation.

We carried out the simulations using the *ns* network simulator [14]. (The simulation scripts are available off the RED-PD Web page [20].) Unless otherwise specified, the capacity of the congested link was 10 Mbps, RED-PD's target RTT R was 40 ms, the packet size was 1000 bytes, and RED was running in packet mode. The Selective Acknowledgement (SACK) [12] version of TCP was used, flows were started at a random time within the first 10 seconds, and aggregated results, where presented, were not taken before 20 seconds into the simulation.

5.1. Probability of Identification

In this section, we explore RED-PD's probability of identifying a TCP flow with a given round-trip time. The identification probability for a CBR flow, studied in [11], is higher than that of a TCP flow with similar sending rate because TCP flows back-off after initial drops, thereby reducing their identification probability. We show a flow's probability of being identified in a single identification round; the eventual throughput of the flow depends on whether the flow is persistently identified.

Figure 9 shows a TCP flow's probability of identification as a function of its sending rate and ambient drop rate at the queue. The simulations were done in a controlled environment where the ambient drop rate at the queue was fixed. The RTT of the TCP flow was varied to get different sending rates. A flow sending at a $f(R,p)$ multiplier of γ had an RTT of $\frac{R}{\gamma}$. The probability is calculated based on 500 seconds of simulation run.

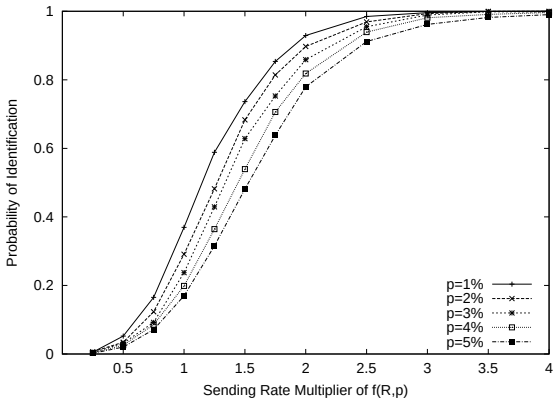

Figure 9. The probability of identification of a TCP flow sending roughly at a rate $\gamma * f(R,p)$.

Figure 9 shows that the identification probability increases quickly as the sending rate of a flow increases. It also shows that a flow sending at less than $f(R,p)$ pkts/s can be identified with some small probability. This occurs when the flow gets unlucky and receives more than its share of packet drops. Because RED is not biased in any way towards a particular flow, a flow sending at less than $f(R,p)$ pkts/s is unlikely to be consistently unlucky in its packet drops. The consequences of a flow getting identified once are not severe; it is monitored with a small initial dropping probability. Monitoring this flow further reduces its chances of being identified again, and thus this flow would soon be unmonitored. High sending rate flows that escape identification in a particular round are identified soon in a near-future round because the identification probability associated with them is high.

5.2. Fairness

This section shows an important property of RED-PD: it is possible to approximate fairness among flows by iteratively increasing and decreasing the pre-filter dropping probability for the high-bandwidth flows. The simulations also show RED-PD's ability to protect the low-bandwidth flows and control the high-bandwidth ones.

The simulation in Figure 10 consists of 11 CBR flows. The sending rate of the first flow is 0.1 Mbps, that of the second flow is 0.5 Mbps, and every subsequent flow sends at a rate 0.5 Mbps higher than the previous flow (the last CBR flow sends at 5 Mbps). Separate lines in Figure 10 show the bandwidth received by each of the 11 CBR flows with RED and with RED-PD, while a third line shows each flow's max-min fair share. The graph shows that with RED, each

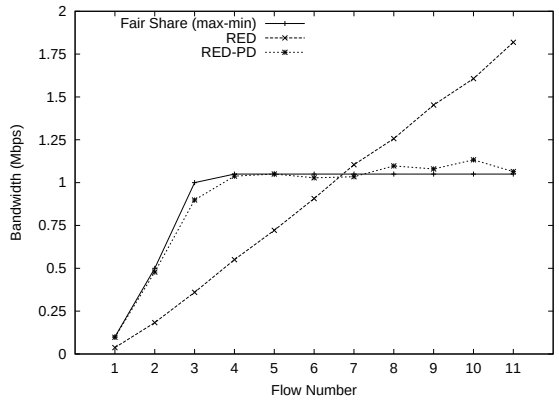

Figure 10. Multiple CBR flows. Flow 1 is sending at 0.1Mbps, flow 2 at 0.5 Mbps and every subsequent flow is sending at a rate 0.5 Mbps more than the previous flow.

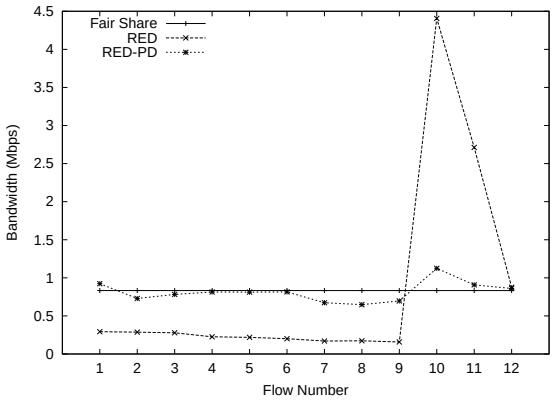

Figure 11. Mix of TCP and CBR flows. Flows 1-9 are TCP flows with RTTs of 30-70 ms. Flow 10, 11 and 12 are CBR flows with sending rates of 5, 3and 1 Mbps respectively.

flow receives a bandwidth share proportional to its sending rate, while with RED-PD all the flows receive roughly their fair share. By concentrating the dropping in the pre-filter for the high-bandwidth flows, RED-PD was able to reduce the ambient drop rate from 63% to about 4%.

The next simulation has a mix of TCP and CBR flows. The aim is to study the effect of high-bandwidth CBR flows on conformant TCP flows and investigate RED-PD's ability to protect the conformant flows. There are 9 TCP flows and 3 CBR flows. The TCP flows have different round-trip times; the first three TCP flows have round-trip times (propagation delay) close to 30 ms (there is some variation in the actual RTTs to preclude synchromization effects), the next three have RTTs around 50 ms, and the last three have RTTs around 70 ms. The CBR flows, with flow numbers 10, 11 and 12, have sending rates of 5 Mbps, 3 Mbps and 1 Mbps respectively. Again, Figure 11 shows the bandwidth of each of the 12 flows with RED and with RED-PD. With RED, the high-bandwidth CBR flows get almost all the bandwidth, leaving little for the TCP flows. In contrast, RED-PD is able to restrict the bandwidth received by the CBR flows to near the target bandwidth (throughput of a hypothetical 40ms TCP flow). Given R of 40 ms, RED-PD monitors not only the CBR flows, but also the TCP flows with RTTs of 30 ms (and occasionally those with 50 ms as well). Each of the CBR flows received a different pre-filter dropping rate, as each flow was successfully restricted to roughly the same throughput.

5.3. Response Time

We now study the response time of RED-PD to sudden changes in a flow's sending rate. Analysis for the time taken to reduce throughput of a high-bandwidth flow and the time to release a flow that cuts its sending rate is done in [11]; we present the results of a simulation here.

Figure 12 shows the results from a simulation with 1 CBR flow and 9 TCP flows. The CBR flow starts with a rate of 0.25 Mbps, increases it to 4 Mbps at $t = 50s$, and decreases it back to 0.25 Mbps at $t = 250s$. The RTT of the TCP flows ranged from 30 to 70 ms.

RED-PD took less than 0.5 seconds to identify and start monitoring the CBR flow, as visible from an immediate decrease in the flow's throughput. The dropping probability of the flow keeps increasing until its throughput reduces to $f(R, p)$ (at about $t = 57s$). Also visible in the figure is the time RED-PD took to unmonitor this flow once it reduced its sending rate at $t = 250s$.

In general, the speed of RED-PD's reaction depends on the ambient drop rate and the arrival rate of the monitored flow, as the probability increase quantum is larger when either of them is higher. This has the desirable effect that if a flow increases its sending rate to a very high level, or leads to a high increase in ambient drop rate, it will be brought down fairly quickly.

5.4. Effect of R, the Target RTT

The simulations in this section illustrate how the choice of RED-PD's configured target RTT R affects both the identification of flows and the bandwidth received by monitored flows. Each column in Figure 13 represents a different simulation, with a different value for R, ranging from 10 ms to 170 ms. In each simulation 14 TCP connections were started, two each with RTTs of 40 ms, 80 ms and 120 ms,

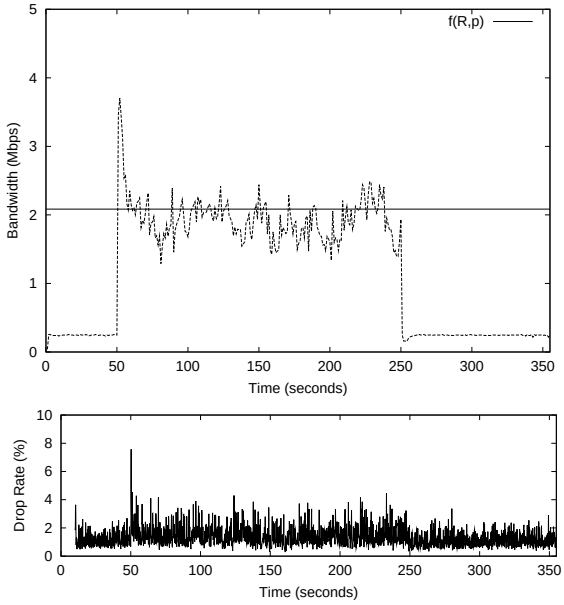

Figure 12. Adapting the dropping probability. The top graph shows the throughput of a CBR flow which changes its sending rate to 4 Mbps at $t = 50s$ and back to 0.25 Mbps at $t = 250s$. The line labeled $f(R, p)$ is based on the ambient drop rate seen over the whole simulation. The bottom graph plots the ambient drop rate over time.

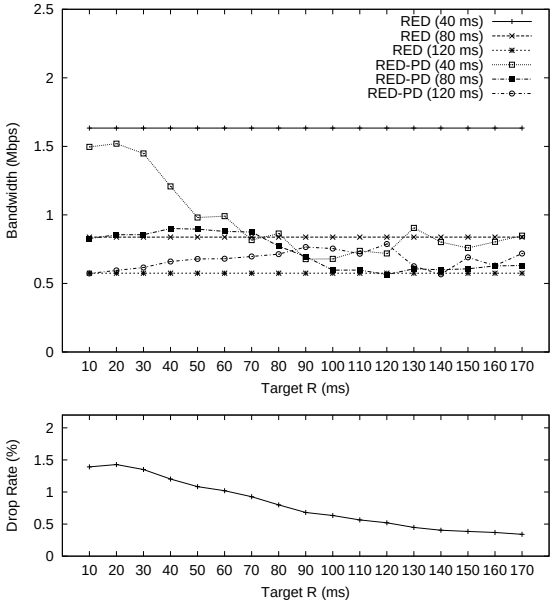

Figure 13. The Effect of Target R. The top graph shows the bandwidth received by 40 ms, 80 ms and 120 ms RTT TCP flows for different values of R. The bottom graph plots the ambient drop rate.

and the rest with RTTs of 160 ms. The top graph of Figure 13 shows the average bandwidth received by the TCP flows with round-trip times from 40-120 ms, while the bottom graph of Figure 13 shows the ambient drop rate. The horizontal lines in Figure 13 show the bandwidth for each traffic type with RED.

For the simulations with R less than 40 ms, RED-PD rarely identifies any flows, and the bandwidth distribution is essentially the same as it would be with RED. However, with R of 40 ms or higher, the TCP flows with 40-ms RTT start to be identified and preferentially dropped. As R is increased, the bandwidth received by the 40-ms TCP flows is decreased, because the target bandwidth, $f(R, p)$, decreases as R increases. In addition, as R is increased the ambient drop rate decreases and the throughput of the unidentified TCP flows increases (as long as their RTT is more than R; 160-ms TCP flows not shown).

As these simulations illustrate, increasing the target RTT R results in more flows being monitored. As R is increased, RED-PD gets closer to full max-min fairness (at the cost of more state). Additionally, increasing R decreases the ambient drop rate and increases the bandwidth available to the unmonitored flows. On the other hand, with a very small value for R, RED-PD has limited impact at the router, and can be used with the goal of controlling only egregiously-misbehaving flows or those conformant flows with very short round-trip times.

Instead of a fixed configured value, R can be varied dynamically as a function of the ambient drop rate or the state available at the router. We intend to explore techniques for dynamically varying R in later work.

6. State Requirements and Complexity

In addition to the state needed by a regular RED queue, RED-PD requires state for the identification engine, which stores the drop history, and the monitored flows. The amount of memory required for the drop history is given by Equation 3, and depends on the target RTT R and the ambient drop rate. An example would help to demonstrate why this should not be a problem. Given $R = 40$ ms and $p = 1\%$, the router needs to store information about packets dropped over the past 1 second. With a total arrival rate of 1 Gbps and an average packet size of 250 bytes this amounts to just 5000 losses ($p = 1\%$) or 200 KB (assuming 40 byte headers). Furthermore, fast memory is not required for storing the drop history because the identification engine does

	State for What	What State	Fast-path Processing	When required
FQ	All flows	Queues	Queue management, scheduling	Packet arrival, departure
FRED	All buffered flows	Count of buffered packets	Drop probability computation, coin tossing	Packet arrival, departure
CSFQ (edges)	All flows	Arrival rate estimate, time of last packet	Update arrival rate estimate, update header, coin tossing	Packet arrival
RED-PD	High-bandwidth flows	Dropping probability, drop history	Coin tossing	Packet arrival

Table 1. A comparison of complexity of some schemes.

not run in the forwarding fast path. Storing drop history as a hash-based data structure would greatly simplify the identification process.

State for the monitored flows includes a classifying data structure used to lookup the dropping probability of a monitored flow (unmonitored flows will be missing from this structure). Lookups matching the forwarding speed can be achieved using sparsely populated hash tables or perfect hash functions. It helps that RED-PD keeps state for only the high-bandwidth flows, which as discussed in Section 3 are a small fraction of the total. A more precise investigation of the state requirements and fairness tradeoffs under various traffic scenarios is a subject of future work.

The complexity of a scheme is not given by the amount of state alone, but is also dependent on the processing done on that state. Table 1 compares RED-PD's complexity with that of several other proposed mechanisms. Missing from the "Fast-path Processing" column is classification, which is common to all the schemes. The "State for What" column gives an idea of the size of the classifying data structure. We see that not only does RED-PD maintain very little state, its fast-path operations are also the simplest.

7. Unresponsive Flows

It is important for schemes that provide differential treatment for flows to provide incentives for end-to-end congestion control by actively punishing misbehaving flows [6]. However, in this work we have addressed the issue of active punishment only briefly.

RED-PD keeps a history of the arrival and drop rates for each monitored flow. A monitored flow is declared *unresponsive* when its arrival rate has not reduced in response to a substantial increase in its drop rate. For flows identified as unresponsive, RED-PD increases the drop probability more quickly, and decreases the drop probability more slowly, to keep the unresponsive flow under tighter control. However, RED-PD does not necessarily reduce the bandwidth obtained by an unresponsive flow, compared with the bandwidth it would have received from RED-PD without having been identified as unresponsive.

RED-PD's test for unresponsiveness can have false positives, in that it could identify some flows that are in fact responsive. The arrival rate of a flow at the router depends not only on the drops at that router, but also on the demand from the application, and the drops elsewhere along the path. In addition, the router does not know the round-trip time of the flow or the other factors (e.g., multicast, equation-based congestion control mechanisms) that affect the timeliness of the flow's response to congestion. The test for unresponsiveness can also have false negatives, in that it might not detect many high-bandwidth flows that are unresponsive.

With its iterative increase and decrease of a flow's drop rate, RED-PD provides an ideal framework for determining the conformance of a flow. Future work will include the investigation of a better unresponsiveness test, and of possibilities for decreasing the throughput for unresponsive monitored flows to significantly less than their fair share, as a concrete incentive towards the use of end-to-end congestion control.

8. Conclusions

We have presented RED-PD, a mechanism that uses drop history to identify high-bandwidth flows, and control their throughput in times of congestion. We have shown that it significantly improves the performance experienced by other flows. In environments like the current Internet, dominated by end-to-end congestion control and in which a small fraction of flows are responsible for a large chunk of bandwidth, RED-PD requires only a small amount of state to do so. Moreover, the fast-path processing in RED-PD is minimal, consisting only of classification over a small fraction of flows, and coin-tossing for monitored flows.

Acknowledgments

We are grateful to Scott Shenker for extensive discussions, and for reviewing an earlier version of this paper. Vern Paxson helped with trace data and analysis. We are thankful to Yin Zhang for discussions that resulted in the multiple list identification approach, Neil Spring for reviewing a draft of this paper, Dina Katabi for helping with the simulations, Kevin Fall for his scripts from [6], and Mark Handley and Jitendra Padhye for TFRC.

References

[1] A. Acharya and A. Rangarajan. Early Regulation of Unresponsive Flows, July 1999. UCSB Tech Report TRCS99-26.

[2] F. Anjum and L. Tassiulas. Fair Bandwidth Sharing among Adaptive and Non-adaptive Flows in the Internet. In *IEEE INFOCOM*, pages pp. 1412– 1420, March 1999.

[3] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *INET*, July 1998.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *ACM SIGCOMM*, September 1989.

[5] W.-C. Feng, D. D. Kandlur, D. Saha, and K. Shin. Blue: A New Class of Active Queue Management Algorithms. Technical Report CSE-TR-387-99, University of Michigan, April 1999.

[6] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, Vol. 7(4):pp. 458–473, August 1999.

[7] S. Floyd, K. Fall, and K. Tieu. Estimating Arrival Rates from the RED Packet Drop History, April 1998. http://www.aciri.org/floyd/end2end-paper.html.

[8] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *ACM SIGCOMM*, August 2000.

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

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

[11] R. Mahajan and S. Floyd. Controlling High-Bandwidth Flows at a Congested Router. Technical Report 01-001, ICSI, April 2001.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, April 1996.

[13] P. E. McKenney. Stochastic Fairness Queuing. In *IEEE INFOCOM*, June 1990.

[14] NS Web Page: http://www.isi.edu/nsnam.

[15] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *IEEE INFOCOM*, March 1999.

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

[17] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate Fairness through Differential Dropping, 2001. Work in progress.

[18] R. Pan, B. Prabhakar, and K. Psounis. CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *IEEE INFOCOM*, March 2000.

[19] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, Jan. 1999.

[20] RED-PD Web Page: http://www.cs.washington.edu/homes/ratul/red-pd/.

[21] M. Shreedhar and G. Varghese. Efficient Fair Queuing using Deficit Round Robin. In *ACM SIGCOMM*, August 1995.

[22] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *ACM SIGCOMM*, September 1998.