RED-PD: Controlling High Bandwidth Flows at the Congested Router*

Ratul Mahajan
http://www.cs.washington.edu/homes/ratul
Department of Computer Science and Engineering
University of Washington, Seattle

April 2, 2001

Abstract

FIFO queueing is simple but does not protect traffic from high bandwidth flows. At the other extreme, full max-min fair techniques provide protection by keeping state for all the flows going through the router, which makes them very complex, especially since most flows through a router are small Web mice. This paper proposes RED-PD (RED with Preferential Dropping) that combines simplicity and protection by keeping state for just the high bandwidth flows. RED-PD uses packet drop history at the router to detect high bandwidth flows in times of congestion, and preferentially drop packets from them. We show that RED-PD increases fairness using significantly less state and the degree of fairness provided by RED-PD can be tuned using a single configurable parameter.

1 Introduction

FIFO queueing is simple to implement, and because it involves no requirements for any uniformity of packet queuing, dropping, and scheduling in the routers along a path, it is well-suited to the heterogeneity and decentralized nature of the Internet. But FIFO provides little protection from high bandwidth flows that consume a lot of bandwidth at the expense of other flows at the router. During times of congestion it is important to control the high bandwidth flows to alleviate the poor throughput experienced by rest of the traffic.

At the other extreme, full max-min fair allocation techniques provide protection by keeping state (some of them even keep separate queues) for all the flows. When most of the flows going through the router are small Web mice (short flows), this is an unnecessarily complex solution, especially for best effort traffic.

In this paper, we present RED-PD (RED [FJ93] with Preferential Dropping), a light weight mechanism that combines the simplicity of FIFO and protection of full max-min fair techniques. RED-PD achieves this by keeping state for the high bandwidth flows only. We call this *partial flow state*.

RED-PD detects high bandwidth flows using the RED drop history. The rationale here is that drops are an unbiased sample of the incoming traffic (at the same time they represent flows that have been sent congestion signal). If a flow has too many drops in the recent history, it is very likely that the flow has a high arrival rate. The amount of drop history used to make this decision depends on the drop rate at the router and a configurable parameter that specifies the *target bandwidth* above which flows should be identified. Flows identified using this process are called *monitored* flows. Packets from a monitored flow are preferentially dropped in a pre-filter before they are inserted into the output queue. The dropping probability is flow specific and is calculated so that the flow's rate into the output queue is limited to the target bandwidth.

The impact of RED-PD on incoming traffic is shown in Figure 1. Assume that flows are identified when their arrival rate is more than the target bandwidth T, and are restricted to T (if there is enough demand from other flows) when monitored. RED-PD has no effect when T is set higher than the maximum arrival rate of a flow. As T is pushed down, bandwidth obtained by the mon-

*This document is a shorter version of the one available at http://www.aciri.org/red-pd. **The work was done jointly with Sally Floyd (floyd@aciri.org)**

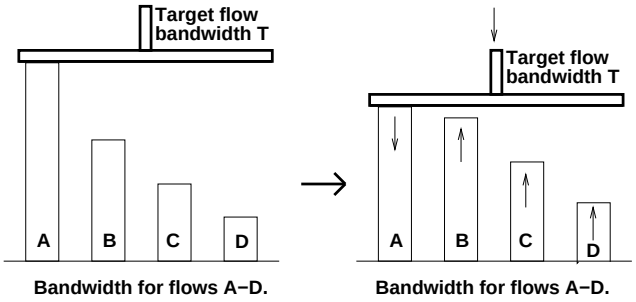

Figure 1: **Restricting flows to a target bandwidth T.**

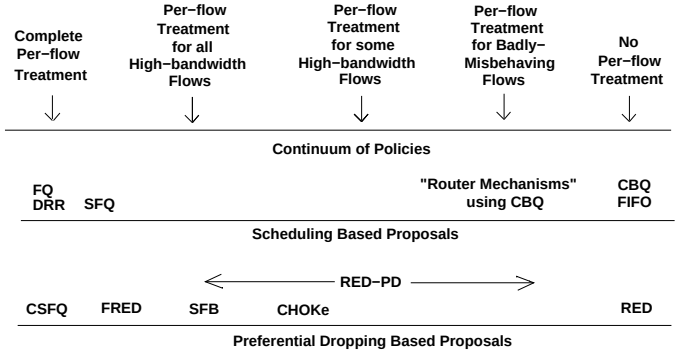

Figure 2: **A continuum of per-flow treatment at the queue.** The top line denotes the range of possible policies; the second and third lines show the rough placement along the continuum of proposals that use scheduling and preferential dropping respectively.

itored flows will be curtailed, which enables the rest of the flows to get more bandwidth because the *ambient* drop rate, which is the drop rate at the output queue, is reduced. For example, when the bandwidth for Flow A is restricted to a lower T in the right portion of Figure 1, it decreases the packet drop rate for the rest of the traffic, and allows Flows B, C and D to receive increased bandwidth at the router.

In the next section we discuss existing proposals based on preferential dropping to enhance fairness among flows. Section 3 provides the results of a trace analysis that confirms that controlling just a few flows gives significant control over bandwidth distribution, which is required for partial per-flow schemes to be effective. Section 4 describe RED-PD in detail. In Section 5, we evaluate RED-PD using analysis and simulations. A discussion of issues related to RED-PD is contained in Section 6, and we conclude in Section 7.

2 Related Work

Techniques to control high-bandwidth flows or enforce fairness among flows can be classified into either scheduling based or preferential dropping based. *Scheduling* approaches place flows in different scheduling partitions (there might be more than one flow in a partition), 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 it's throughput. Scheduling mechanisms offer more precise control, while some preferential dropping mechanisms are fair only in the probabilistic sense. However, scheduling mechanisms generally have higher state requirements. Preferential dropping based schemes can easily incorporate other desirable qualities like permitting small bursts to go through, active queue management, and active punishment of misbehaving flows. Figure 2 classifies the existing approaches based 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 this section, we provide a comparison with preferential dropping based approaches only because they are closer to our approach.

Like RED-PD, Core-Stateless Fair Queuing (CSFQ) [SSZ98] and Flow Random Early Detection (FRED) [LM97] use preferential dropping in concert with FIFO scheduling. FRED maintains complete per-flow state at the router for all flows that have packets in the queue. The dropping probability of a flow depends on the number of buffered packets. FRED's fair allocation of buffers can yield very different fairness properties from fair allocation of bandwidth [SSZ98], as a result of which the fairness properties of FRED are not predictable. In contrast, RED-PD's effect on incoming traffic is well understood, independent of arrival pattern of packets.

CSFQ approximates fair queuing without using per-flow state in the core of an *island* of routers. The edge routers, which maintain per-flow state, estimate the sending rate of a flow and insert the estimate in its header which is used by the core routers to preferentially drop from them based on the flow's fair share estimate. Unlike CSFQ, RED-PD does not require modification to

packet headers and a RED-PD router can operate in isolation without the cooperation of other routers making deployment much simpler.

In CHOKe [PPP00] an incoming packet is matched against a random packet in the queue, and both packets are dropped if they belong to the same flow. Otherwise the incoming packet is admitted with a certain probability. The rationale being 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 (compared to the buffer space) and even the high-bandwidth flows have very few packets in the queue. The simulations in the paper show that CHOKe achieves limited performance in controlling high bandwidth CBR flows, which run away with much more their fair share.

A more recent approach in [SBPP01] is an outgrowth of 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. This is a simultaneous but independent work that is still unpublished and hence not possible for us to comment on its details (our comments are based on discussions with one of the authors).

Stochastic Fair Blue (SFB) [FKSS99] relies on multiple levels of hashing to identify high bandwidth flows. The scheme works well only in presence of a few high-bandwidth flows. With multiple high bandwidth flows SFB ends up punishing even the low bandwidth flows as more bins get polluted. On the other hand, when RED-PD is state limited, it would not punish a fragile low bandwidth flow; it would drop from as many high bandwidth flows as its state limitations permit.

Floyd and Fall in [FF99] discuss mechanisms for identifying high-bandwidth flows from the RED drop history, and using CBQ based mechanisms to partition misbehaving and conformant flows into different classes. They did not present a complete solution, and the approach is limited by the choice of an aggregate scheduling-based mechanisms instead of the per-flow preferential dropping mechanisms used in RED-PD.

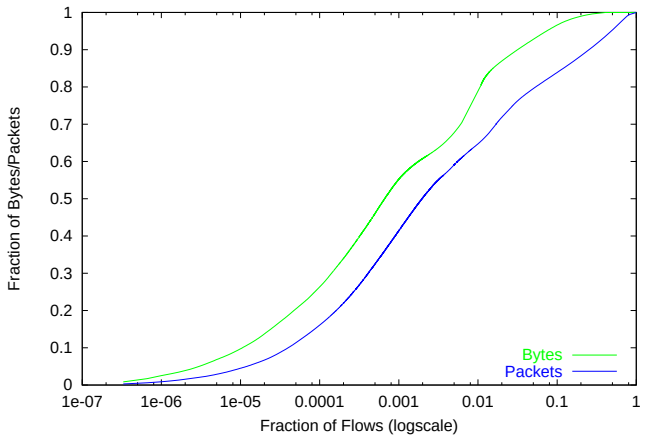

Figure 3: **Skewedness of bandwidth distribution**

3 Why Partial Flow State Approach Works?

In this section we present trace results that justify our belief that approaches like RED-PD that keep state for only high bandwidth flows can be effective. For these approaches to work, it is crucial that a small fraction of flows be responsible for a large chunk of bandwidth. Given this skewed distribution of bandwidth, controlling the bandwidth obtained by this fraction gives significant control to the router and enables it to bring down the drop rate at the router, that in turn leads to higher throughput for other flows.

The traces we examined indeed show that a small fraction of flows are responsible for a large fraction of the bytes (this result has been confirmed by various other previous studies, e.g. [CMT98]). We further show that identifying and preferentially dropping from this small number of flows can be a powerful tool.

Figure 3 shows results from a one-hour-long trace taken from UCB DMZ in August 2000. The graph 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. About 96% of the bytes and 84% of the packets came from just 10% of the flows.

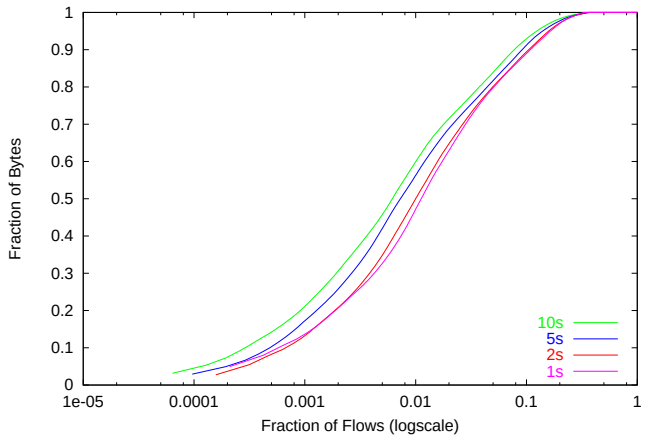

Figure 4: **Skewedness over smaller time scales**

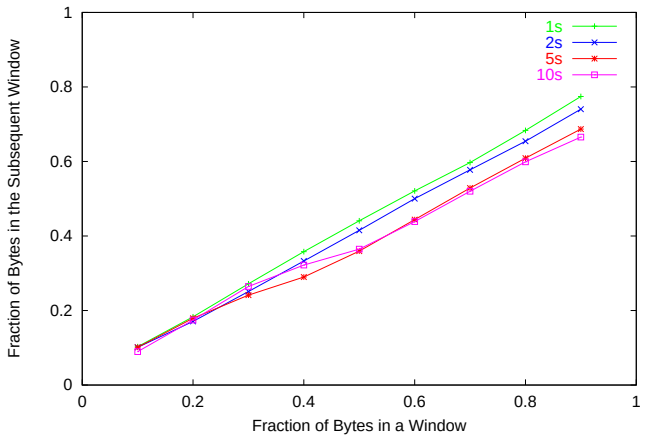

Figure 5: **Predictive nature of bandwidth consumption**

RED-PD's identification is based on bandwidth and not on total bytes sent over the lifetime of the flow. Thus, for effectiveness of our approach, it should be case that a small fraction of flows send most of the bytes even on shorter time scales. The graph in Figure 4 plots this information for smaller time windows. It shows the fraction of flows responsible for a given fraction of bytes. We can see that the skewedness holds not only for long time periods but also for smaller time windows, scales at which identification-based fairness approaches are likely to identify the high-bandwidth flows.

Another important property for an identification based approach to be successful is that the identified high bandwidth flows in a given interval should be a good predictor of the high bandwidth flows in the succeeding interval.

Figure 5 plots the fraction of bandwidth consumed in the subsequent interval by flows which accounted for a particular amount of bandwidth (x-axis) in the current interval. For example, from the graph in Figure 4 we can see that in a time window of 5 seconds, 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 window. If an identification-based scheme were to identify and restrict just these small fraction of flows, it would save a fair amount of bandwidth for other flows.

4 RED-PD

There are two components in RED-PD: identifying the high bandwidth flows, and controlling the bandwidth obtained by these flows. The next two subsections discuss each of them respectively.

4.1 Identifying High Bandwidth Flows

RED [FJ93] (Random Early Detection) is a gateway algorithm that randomly drops (or marks) packets on detecting incipient congestion. Since RED drops are probabilistic, and not result of a buffer overflow, they can be taken as random samples of the incoming traffic. RED-PD uses the RED drop history to identify the flows sending more than the configured target bandwidth. Unlike what the name RED-PD implies, our technique can be used with any active queue management scheme that distributes drops randomly.

Drop history is a cheap means for identification. It is a nearly random sample of incoming traffic. A flow with more drops in the history is very likely to have a higher sending rate. At the same time, it represents flows that have been sent congestion signals. Thus, we use the packet loss samples to roughly estimate the arrival rate of the flow and to confirm that the identified flow has in fact received loss events.

Instead of keeping the drop history as a single big list, RED-PD partitions the history into multiple lists con-

taining drops from consecutive intervals of time. The identified high bandwidth flows are the ones occurring regularly in the these lists. The time interval over which a list is compiled depends on the drop rate at the queue and target bandwidth. Next, we describe how this interval and definition of "regular" is chosen to restrict identification to flows above the target bandwidth.

The target bandwidth is the bandwidth obtained by a TCP flow of target RTT R with the drop rate p at the queue. Choosing target RTT R is different from choosing an absolute target bandwidth T. While bandwidth obtained by a TCP flow depends on the drop rate, T would have been independent of it. It is easier to understand the impact of RED-PD with a target RTT in an environment dominated by TCP or TCP compatible congestion control.

Let $f(r, p)$ denote the average sending rate in pkts/s of a TCP flow with a round-trip time r and a steady-state packet drop rate p. From the deterministic model of TCP in [FF99], we have:

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

Reasons for choosing this equation instead of the more precise one given in [PFTK98] are discussed in Appendix A.

RED-PD's goal is to identify flows that are sending at a rate higher than $f(R, p)$, the sending rate of the reference TCP. In the deterministic model, a TCP flow has a single drop in every congestion epoch, which contains $\frac{1}{p}$ packets. Hence, the *congestion epoch length* of a TCP flow with RTT r and drop rate p is

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

Flows sending at a rate higher than $f(R, p)$ will have, on average, more than one drop in $CL(R, p)$ seconds. For accuracy, we restrict identification to flows with more than K loss events in $K \times CL(R, p)$ seconds. A loss event is one or more losses in a window of data. To count loss events instead of losses, RED-PD maintains the drop history of $K \times CL(R, p)$ seconds in terms of M separate smaller lists. Flows with drops in K or more lists out of the M are identified. Losses occurring in the same list are considered as occurring in a window of data, while those in different lists as loss events. (Appendix B shows the advantage of using multiple lists over a single big list). Thus, the length of each list is

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

To determine the length of a list, the router estimates the current packet drop rate at the queue. The router can directly measure its loss rate as the number of drops divided by the number of arrivals. We use exponential averaging to smooth the drop rate estimate over successive intervals.

4.1.1 Choosing Identification Parameters

We now discuss our choice of various parameters used in the identification scheme. We first consider K, the number of $CL(R, p)$s that make up the drop history. Larger values of K make identification more reliable and make it more likely that the flow's arrival rate reflects the response of end-to-end congestion control to the packet losses received during that period. These benefits come at the expense of an increase in the time required to identify high-bandwidth flows. Smaller values of K increase the chances of catching unlucky flows, that is, low-bandwidth flows that happen to suffer more drops. Since we use the absence of a drop in all the lists as a criterion for decreasing the monitored flow's dropping probability (Section 4.2), a small value of K would lead to frequent changes in the dropping probability. In our simulations, a value of $K = 3$ gives a reasonably prompt response along with a reasonable protection for unlucky flows.

The next parameter to consider is M, the number of separate lists. Clearly, M should be greater than K because we identify flows that have drops in at least K lists. To count drops in the same window as a single loss event, the drop-list interval should be longer than the typical round trip time of flows in the queue. Given our choice of $K = 3$, a value of $M = 5$ has worked well in our simulations.

The choice of R, the target round trip time, will vary at different installations depending on factors like the composition of traffic, state limitations and desired degree of fairness. Increasing R increases the degree of fairness (as more flows are identified), and at the same time increases the state kept at the router. Effects and guidelines of choosing a particular R are discussed in detail in Sections 5.4 and 6.3.

4.2 Preferential Dropping

After identifying high bandwidth flows, we need to reduce the arrival rate of these flows to the output queue. The control mechanism should have the following properties:

- It should be light-weight and FIFO compatible.
- It should not only protect other traffic from the monitored flows, but also provide relative fairness among the monitored flows, (lumping all monitored flows in one aggregate lacks this property).
- The monitored flows should not be starved, and at the same time, not given more bandwidth than they would have obtained in the absence of monitoring. This rules out solutions that give "leftover bandwidth" to the monitored flows, or that give a fixed amount of bandwidth to a monitored flow without regard to the level of unmonitored traffic.

We now describe a technique, per-flow preferential dropping, which has all the above desired properties. Preferential dropping places a *pre-filter* in front of the output queue. Packets from monitored high-bandwidth flows pass through this pre-filter where they are dropped with a certain probability, and the survivors are put in the output queue. The unmonitored traffic is put in the output queue directly. Figure 6 shows the architecture of a RED-PD router.

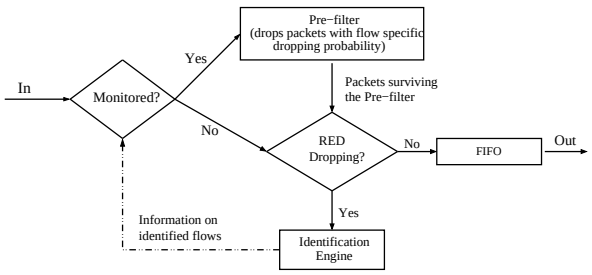

Figure 6: **Architecture of a RED-PD router.**

This simple mechanism provides relative fairness between monitored flows when a high-bandwidth flow is dropped in proportion to its excess sending rate. RED-PD does not protect the monitored flows from the general congestion at the link, because the output queue does not differentiate between the flows once the pre-filter has cut down on the monitored traffic. So the packets from the monitored flows can be dropped at the output queue too if it is congested.

If RED-PD is monitoring most of the bandwidth coming to the router, there is a danger of link under-utilization, because despite of low demand at the output queue, packets may be dropped in the pre-filter. To avoid this, the pre-filter does not drop packets from monitored flows when there is insufficient demand (as measured by RED's average queue size) from the unmonitored traffic.

4.2.1 Computing Dropping Probability

The most important component of the control scheme is determining the dropping probability of each monitored flow. In an ideal setting we could measure each monitored flow's arrival rate and compute dropping probability such that it reduces the rate to output queue to $f(R, p)$. In RED-PD, instead of determining the dropping probability for a monitored flow from an estimated arrival rate, we base the dropping probability directly on the identification mechanism itself.

RED-PD slowly increases the pre-filter dropping probability for a monitored flow until it is no longer identified as high bandwidth in the output queue. This implicitly uses $f(R, p)$ as the yardstick for increasing (or decreasing) a flow's pre-filter dropping probability. That is, if a monitored flow continues to be identified from its packet drops in the output queue, its dropping probability in the pre-filter is increased, and, when a monitored flow's packet drops in the output queue drop below a certain level, its dropping probability in the pre-filter is decreased.

When a flow is identified for the first time, RED-PD starts monitoring it, and packets from it are dropped in the pre-filter with a small initial dropping probability. The identification process considers drops at the entry to the output queue only, not in the pre-filter. Thus, the identification process is concerned with the flow's arrival rate to the output queue, not to the router itself; the two quantities would be different for a monitored flow.

```
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 list)
  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 probability reduction**

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

foreach f  (flows that appear in K
           or more 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 probability increase**

A monitored flow will be identified again by the identification process if its arrival rate to the output queue is higher than $f(R,p)$. For such flows the dropping probability is increased. 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. Once the dropping probability of a flow reaches a negligible value, the flow is unmonitored altogether. With this iterative increase and decrease, the router settles around the right pre-filter dropping probability for a monitored flow.

The dropping probability for a monitored flow is not changed when the the flow appears in at least one but fewer than K of M drop lists. This provides the necessary hysteresis for stabilizing the dropping probability. In addition, 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.

We now specify how the router changes a flow's dropping probability. The pseudocode for probability reduction is given in Figure 7. The dropping probability is decreased by simply halving it; though the reduction is bounded by a maximum allowable decrease in one step, to reduce oscillations. These oscillations could be a result of one or both, the imprecision of using packet drop history as arrival rate estimate, and the reaction of the flow's end-to-end congestion control mechanisms to decrease drop rate. The absence of the flow in all the drop-lists can be the result of it getting lucky and not the flow's reduction in sending rate. In such cases the upper bound ensures that control over a flow being monitored with high dropping probability is not loosened by a large amount in one step.

When increasing a flow's pre-filter dropping probability the ambient drop rate and the arrival rate of the flow need to be considered. When the ambient drop rate is high, the high bandwidth flows need to be brought down sooner, so the increase quanta should be large. In addition, different monitored flows will have different arrival rates to the output queue, which is also reflected by different number of drops suffered by them. Increase quanta for flows with higher arrival rate to the output queue should be larger.

Figure 8 shows the pseudocode for probability increase step. At a given instant we have a group of identified flows whose dropping probabilities have to be increased. Let the drop rate in the output queue be p, and the average number of drops among the flows identified in this round be avg_drop_count. A possible method for deciding a flow's increase quanta that takes into account both the ambient packet drop rate and the relative sending rate of the different monitored flows is:

$$P_{delta} = (drop_f/avg\_drop\_count) * p \qquad (4)$$

where $drop_f$ is the number of drops for flow f. If this increase quantum is more than the flow's existing drop

rate, we just double the flow's dropping probability (to make sure we don't increase a flow's drop rate all of a sudden). The existing drop rate for a flow is the sum of the drop rate at the pre-filter (zero for unmonitored flows) and the drop rate at the output queue.

5 Evaluation

We use a combination of analysis and simulation to evaluate RED-PD for several properties. Since identification is the first step in controlling a high bandwidth flow, in §5.1 we study RED-PD's effectiveness in identifying a high bandwidth flow. Fairness is a crucial property for congestion control schemes. RED-PD's ability to enforce fairness using the iterative probability increase and decrease is investigated in §5.2. It is important that RED-PD react in a reasonable amount of time 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 R effects the degree of fairness and amount of state kept by the router. More simulation results are presented in Appendix E.

We carried out the simulations using the NS network simulator [NS]. 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. Selective acknowledgement (SACK) [MMFR96] version of TCP was used, flows were started at a random time within 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 RTT. Identification probability for CBR flow is analyzed in Appendix C. 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 and the probability change mechanism.

A TCP flow's probability of identification as a function of its sending rate and ambient drop rate at the queue is plotted in Figure 9. Simulations were done in a controlled environment where the ambient drop rate at the queue was fixed. RTT was varied to get flows sending at different rates. For example, given that the the target RTT R was 40 ms, the RTT of the TCP flow with sending rate $0.5f(R,p)$ was 80 ms and of that with rate $2.0f(R,p)$ was 20 ms.

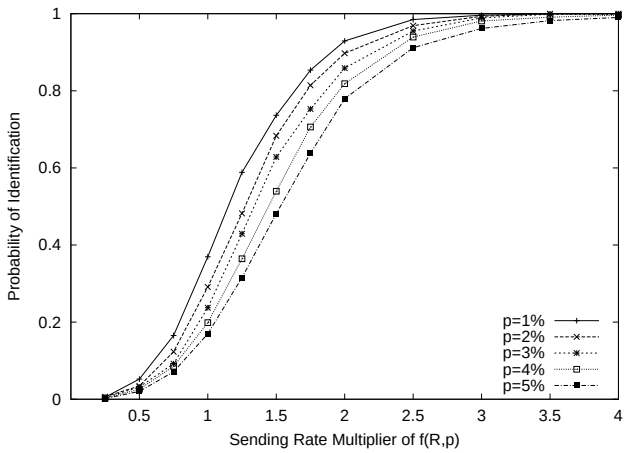

Figure 9: **The probability of identification of a flow sending at a rate $\gamma * f(R,p)$ for $K/M = 3/5$.**

The decrease in identification probability with an increase in drop rate occurs because as the ambient drop rate increases, Equation 1 overestimates the sending rate of a TCP flow.

Figure 9 shows that identification probability for flows sending below $f(R,p)$ pkts/s is non-zero. Identification of flows below $f(R,p)$ is not harmful as long as the identification process is fair. It should not be the case that a particular flow sending below $f(R,p)$ is consistently identified while others flows, that send at a similar rate, go scot-free. We rely on RED, which is not biased towards any flow, to fairly distribute the drops and prevent this. In such a situation the graph just tells us that identification (and preferentially dropping) will also happen for flows sending a little below $f(R,p)$. The figure also shows that a flow sending at more than $f(R,p)$ may escape in a particular round. It is not a concern as this flow would be identified soon, in a round in very near future.

5.2 Fairness

This section shows an important property of RED-PD: it is possible to approximate fairness among flows by

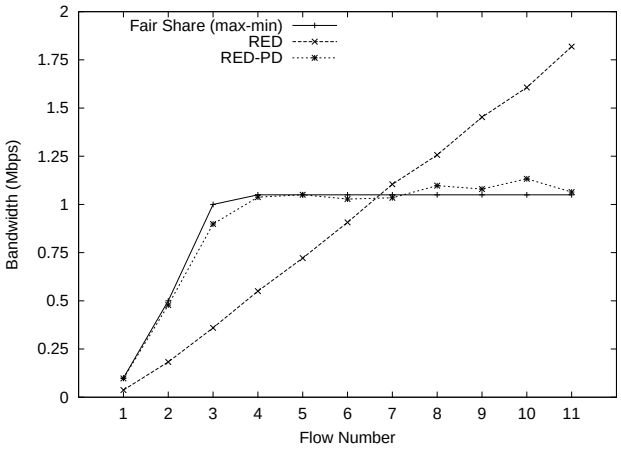

Figure 10: **Simulation with multiple CBR flows.** Flow 1 is sending at 0.1Mbps, flow 2 at 0.5 Mbps and every subsequent flow is sending at a rate 0.5 Mbps more than the previous flow.

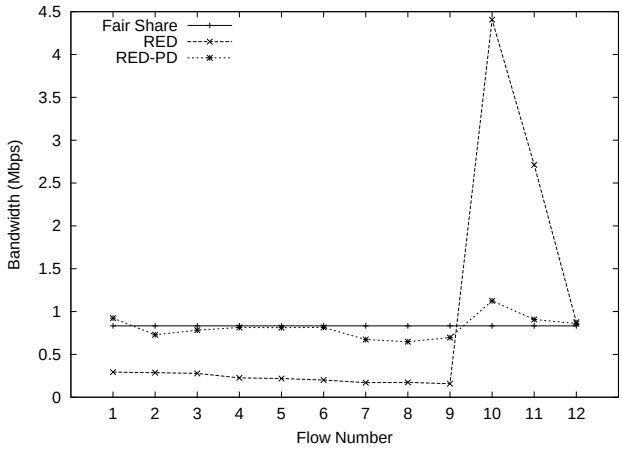

Figure 11: **Simulation with a mix of TCP and CBR flows.** Flows 1-9 are TCP flows with RTTs of 30-70 ms. Flow 10, 11 and 12 are CBR flows with sending rates of 5, 3and 1 Mbps respectively.

iteratively increasing and decreasing pre-filter dropping probability. 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 of different rates. 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 flow receives a bandwidth share proportional to its sending rate, while with RED-PD all the flows receive roughly their fair share. Without RED-PD, the ambient drop rate is about 63%, while with RED-PD the ambient drop rate is reduced to roughly 4% by concentrating the dropping to high bandwidth flows.

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 them. 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 close to 30 ms (there is some variation in the actual RTTs), 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 their fair share. With a target R of 40 ms, RED-PD monitors not only the CBR flows, but also TCP flows with RTTs 30 ms (and occasionally those with 50 ms as well). Each of the CBR flows received a different pre-filter dropping rate, as each CBR flow was successfully restricted to roughly its fair share.

5.3 Response Time

This section is devoted to studying the response time of RED-PD to sudden increase or decrease of a flow's sending rate. A detailed analysis is presented in Appendix D. We present the results and verify them using simulations here.

Assume that a flow suddenly starts sending at $\gamma * f(R, p)$ pkts/s. Under the assumption that the ambient drop rate at the router is independent of the sending rate of a single flow, which would be true for places with high degree of statistical multiplexing, the time taken by RED-

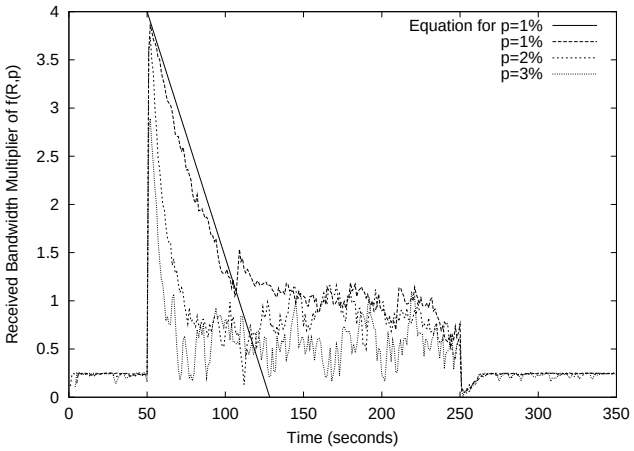

Figure 12: **The response time of RED-PD.** The CBR source increases its sending rate to $4*f(R,p)$ at $t = 50s$ and reduces it back to $0.25 * f(R,p)$ at $t = 250s$.

PD to clamp this flow to $\alpha * f(R,p)$ pkts/s is:

$$t_{cutdown} = \frac{(\gamma - \alpha)RK(M-1)}{\gamma p\sqrt{1.5p}M} \qquad (5)$$

Figure 12 shows a simple simulation to test Equation 5. A CBR source passed through a queue with a fixed loss rate. The CBR source initially sends at $0.25 * f(R,p)$ and increases its sending rate to $4 * f(R,p)$ ($\gamma = 4$) at $t = 50s$. The line marked "Equation" is based on Equation 5, and the rest of the lines are simulation results for received bandwidth averaged over 1-second intervals. It can be seen that the equation predicts the simulation results very closely.

Equation 9 in Appendix D.2 gives the time to release a flow that reduces its sending rate. For the simulation in Figure 12, $P = 0.75$ (the sending rate is $4 * f(R,p)$), the release time for $p = 1\%$ comes out to be 10.58 seconds (with p_d=0.05, $n = 13$ and $m = 5$) which is very close to what we see in simulation (immediately after the sending rate cut-down at $t = 250s$).

The above simulation was done in a controlled environment where the output queue had a constant configured drop rate to prove the validity of the analysis under the assumptions made by it. Figure 13 shows the results from a normal simulation with one CBR flow and 9 TCP flows over a 10 Mbps link. The CBR flow was started with the initial rate of 0.25 Mbps. At $t = 50s$ the CBR flow increases its sending rate to 4 Mbps, and

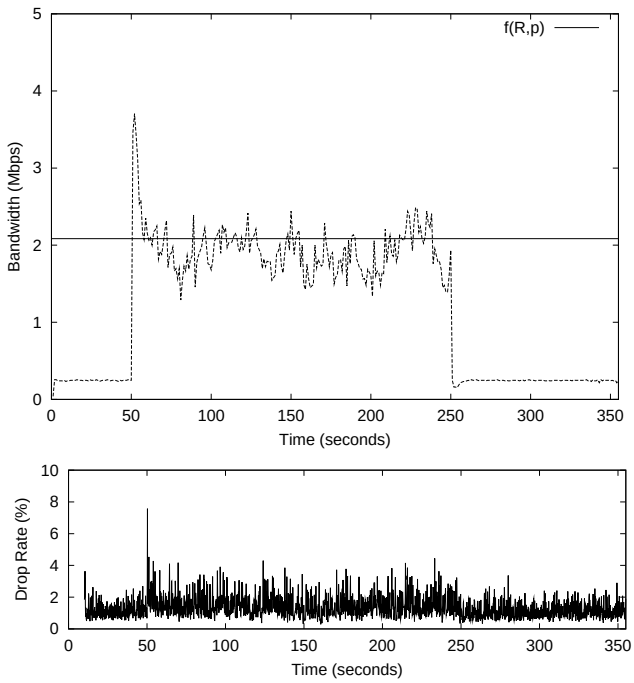

Figure 13: **Adapting dropping probability.** The top graph shows the throughput of a CBR flow which changes its sending rate to 4 Mbps at $t = 50s$ and back to 0.25 Mbps at $t = 250s$. The line labeled $f(R,p)$ is based on the ambient drop rate seen over the whole simulation. The bottom graph plots the variation of ambient drop rate with time.

at $t = 250s$ it decreases its sending rate back to 0.25 Mbps. The RTT of TCP flows ranged from 30 to 70 ms.

In this simulation, it takes just 7 seconds to bring down the throughput of the CBR flow. The big difference from the analysis comes from the fact that, in the simulation, sudden increase in the sending rate of the CBR flow leads to an increased drop rate at the router (visible in the lower graph). Thus, if a flow's increased rate also increases the drop rate at the router, the flow would be brought down much sooner. If it does not change the drop rate significantly, a fast reaction is not critical in the first place.

The speed of RED-PD's reaction depends on both the ambient drop rate and the arrival rate of the monitored flow. If the ambient drop rate is high or arrival rate is higher than other monitored flows, the increase quanta is large and the dropping probability is increased faster.

So, if a flow increases its sending rate to huge levels, it would be brought down more quickly initially, and the rate of bringing it down would slow down as bandwidth received by it goes down. This effect can be seen in the graph (slope before 2.5 on the y-axis is higher than the slope after it).

5.4 Effect of R, the Target RTT

The simulations in this section illustrate how the choice of RED-PD's configured RTT parameter R affects the identification of flows for monitoring as well as the bandwidth received by monitored flows. Each column in Figure 14 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, and the rest with RTTs of 160 ms. The top graph of Figure 14 shows the average bandwidth received by the TCP flows with round-trip times from 40-120 ms, while the bottom graph of Figure 14 shows the ambient drop rate. The horizontal lines in Figure 14 show the bandwidth for each traffic type with RED.

For the simulations with R less than 40 ms, the smallest RTT in the traffic, RED-PD rarely identifies any flows, and the bandwidth distribution is essentially the same as it would be with RED. However, for the simulations with R of 40 ms or higher, the short TCP flows with 40-ms RTTs start to be identified and preferentially dropped. Note that as R is increased, the bandwidth received by the short TCP flows is decreased, because the target bandwidth for a monitored flow is $f(R, p)$, which decreases as R increases. In addition, as R is increased the ambient drop rate decreases and the throughput for the long TCP flows increases (though this is not shown in Figure 14).

As these simulations illustrate, increasing RED-PD's configured value of R results in more flows being monitored, and more drops occurring in the pre-filter for the monitored flows. Thus, as R is increased, RED-PD gets closer to full max-min fairness. In addition, increasing R decreases the ambient drop rate, and therefore increases the bandwidth available to web mice and other unmonitored flows. The simulations also show that, 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

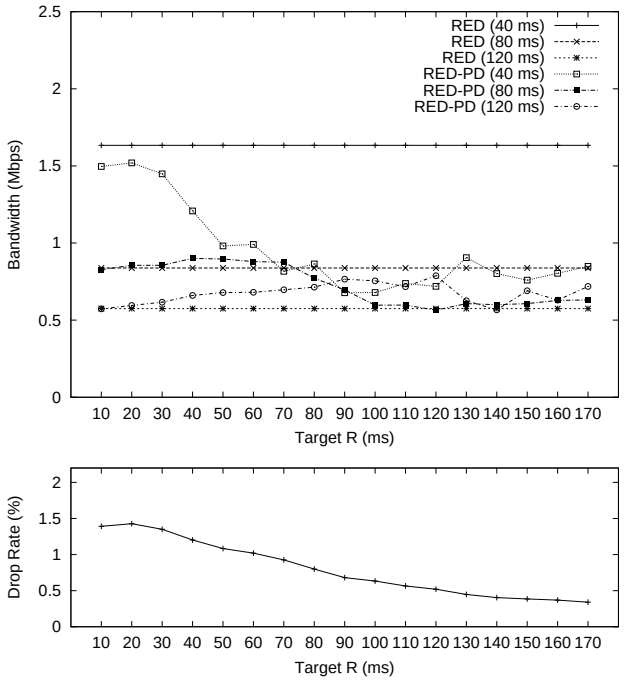

Figure 14: **The Effect of Target R.** The top graph shows the bandwidth received by 40 ms, 80 ms and 120 ms RTT TCP flows for different values of R. The bottom graph plots the ambient drop rate.

conformant flows with very short round-trip times.

5.5 Additional Simulations

The following simulations have been included as Appendix E due to space considerations.

- §E.1: Throughput of monitored flows during persistent ambient drop rate;
- §E.2: Simulation involving Web traffic.
- §E.3: Effect of RED-PD on a mix of TCP and TFRC [FHPW00] traffic.
- §E.4: RED-PD's effect on flows traversing multiple congested links;
- §E.5: Byte mode operation.

All of them have yielded results that show the effectiveness of RED-PD for the properties it was evaluated for.

6 Discussion

This section discusses some issues pertaining to RED-PD. We talk about the fairness properties, state requirements, choosing target RTT, and dealing with unresponsive and evasive flows.

6.1 Fairness Properties

RED-PD provides limited max-min fairness, in which it controls the bandwidth allocated to the high bandwidth flows. In environments where all of the high-bandwidth flows are conformant and have round-trip times considerably larger than the target RTT R, RED-PD has no impact. However, in environments with high bandwidth non-conformant flows or with flows with round-trip times less than R, RED-PD changes the bandwidth allocation of the underlying system by restricting the throughput of high-bandwidth flows. RED-PD's control of the throughput of high bandwidth flows to the output queue is accompanied by a reduction in the ambient drop rate, which is the drop rate seen by unmonitored flows. A decrease in the ambient drop rate results in an increase in the bandwidth received by unmonitored flows.

6.2 State Requirements

In addition to the state needed by a regular RED queue, RED-PD requires state for the identification engine and monitored flows. The identification engine stores M drop lists. The amount of memory required depends on the target RTT R, the ambient drop rate and the number of flows competing at the queue. For example, with $R = 40$ ms, and $p = 1\%$, the router needs to store information about packets dropped over the past 1 second, which should not be a problem even for high-speed routers. It should also be noted that fast memory is not required for storing drop lists as the identification engine does not run in the forwarding fast path. In rare cases when the router does not have enough memory to store the headers for all the drops, it can either randomly sample the drops, or, sort and store just the drops from the high senders (as represented by the flows with the most drops) when retiring the current list to start a new one. This would restrict identification to just the highest-bandwidth flows, and should not lead to any major changes in the behavior of RED-PD.

Apart from the drop history, RED-PD keeps state for monitored flows. Upon packet arrival, the router determines if it belongs to a monitored flow, and applies the appropriate preferential dropping to the packet before adding it to the output queue. Lookups matching the forwarding speed can be achieved using sparsely populated hash tables or perfect hash functions. It helps that RED-PD does not keep state for all the flows going over the link. A quick look at Figure 4 tells us that only 10% of the flows accounted for more than 80% of the link bandwidth. In this situation, by keeping state for only 10% of the flows RED-PD would be very effective in controlling the bandwidth distribution on the link. Detailed investigation of state requirements and fairness tradeoffs involved in RED-PD under various traffic scenarios is a subject of future work.

6.3 Choosing R, the Target RTT

As was illustrated by the simulations in Section 5.4, the choice of the target round-trip time R determines RED-PD's operating point along the continuum of greater or lesser per-flow treatment at the congested queue. A larger value for R results in greater per-flow treatment, requiring more state at the router, and coming closer to full max-min fairness. In contrast, a smaller value for R leads us to the opposite end of the spectrum.

Instead of a fixed, configured value for R, another possibility is to vary R dynamically, as a function of the ambient drop rate and/or of the state available at the router. We intend to explore techniques for dynamically varying R in later work.

6.4 Unresponsive Flows

It is important for schemes that provide differential treatment for flows to provide incentives for end to end congestion control. RED-PD keeps a history of arrival and drop rate for each monitored flow. A monitored flow is declared unresponsive when its arrival rate has not reduced in response to a substantial increase in drop rate. An unresponsive flow's probability increase quanta are bigger and decrease quanta are smaller, to keep it under tighter control (this does not necessarily reduce the bandwidth obtained by an unresponsive flow compared

with bandwidth it received when no unresponsiveness test was in place).

The test above is lapse in that it does not enforce a TCP-compatible behavior, which is difficult because of possibilities like limited application level demand, drops at some other router and an unknown RTT may not present a TCP equation like response at the congested router. 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, will be addressed in future work.

6.5 Evasive Flows

Given a complete knowledge of the RED-PD identification mechanisms at the router, a high-bandwidth flow could possibly evade the identification procedure by restricting its sending bursts to at most $K - 1$ of M intervals. A flow is not likely to be able to do this without a precise knowledge of length and start times of the identification intervals, which are not fixed but change with the drop rate at the router. However, it is true that the more bursty the sending pattern of a flow over successive identification intervals, the less likely it is to be detected by the identification mechanism. To protect against bursty flows, identification should extend to flows that receive drops in less than $K - 1$ of M lists, but have a very high number of packet drops in these intervals.

7 Conclusions

We have presented RED-PD, a partial per-flow state mechanism that uses packet drop history at the router to detect high bandwidth flows in times of congestion, and preferentially drops packets from them. We have shown, in a range of environments, that the proposed mechanism successfully controls the bandwidth obtained by high bandwidth flows and increases the fairness among flows using significantly less state. We also showed that RED-PD has a reasonable response time to sudden changes in a flow's arrival rate. It reacts faster when the flow is consuming too much bandwidth or the drop rate at the router is high. The level of fairness provided by RED-PD can be controlled using the target RTT R, and we showed how it varies with the choice of R.

Acknowledgments

I am grateful to Sally Floyd, my collaborator on this project, and without whose immense contribution this work was not possible. Thanks are also due to my advisor, David Wetherall, who has been very supportive throughout and provided useful feedback. I would like to thank Scott Shenker for extensive discussions, and for reviewing an earlier version of this paper. Vern Paxson helped with trace data and analysis. I am thankful to Yin Zhang for discussions that resulted in the multiple list identification approach, Neil Spring for reviewing the paper, Dina Katabi for helping with the simulations, Kevin Fall for his scripts from [FF99], and Mark Handley and Jitendra Padhye for TFRC.

References

[CMT98] Kim Claffy, Greg Miller, and Kevin Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *INET*, July 1998.

[FF99] Sally Floyd and Kevin 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.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-Based Congestion Control for Unicast Applications. In *ACM SIGCOMM*, August 2000.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, Vol. 1(4):pp. 397–413, August 1993.

[FKSS99] Wu-Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang Shin. Blue: A New Class of Active Queue Management Algorithms. Technical Report CSE-TR-387-99, University of Michigan, April 1999.

[LM97] Dong Lin and Robert Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM*, September 1997.

[MMFR96] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options. RFC 2018, April 1996.

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

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, August 1998.

[PPP00] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *IEEE INFOCOM*, March 2000.

[SBPP01] Scott Shenker, Lee Breslau, Rong Pan, and Balaji Prabhakar. Approximate Fairness through Differential Dropping, 2001. Work in progress.

[SSZ98] Ion Stoica, Scott Shenker, and Hui Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *ACM SIGCOMM*, September 1998.

A Choice of TCP Response Equation

Instead of Equation 1, we could also use the equation $f_1(r,p)$ given in [PFTK98],

$$f_1(r,p) = \frac{1}{r\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \quad (6)$$

for TCP retransmit timeout value t_{RTO}, which can be approximated as $4r$. This equation incorporates the effects of retransmit timeouts, and is based on a model of Reno TCP experiencing independent packet drops. While the TCP throughput equation $f_1(R,p)$ more accurately models TCP behavior, it basically gives the long-term sending rate of a TCP connection. A conformant TCP flow that has not suffered a retransmit timeout in the most recent several congestion epochs might send at a rate higher than $f_1(R,p)$ over that period. The equation for $f(R,p)$ is closer to the sending rate of the TCP flow over the short term (of several congestion epochs) in the absence of retransmit timeouts. For low to moderate levels of congestion, $f(R,p)$ and $f_1(R,p)$ give similar results, and the difference is negligible. However, for higher packet drop rates, when a congestion epoch is quite short, a flow could easily go for several epochs without receiving a retransmit timeout, and in this case it would seem desirable to use $f(R,p)$ to be properly conservative in our identification of high-bandwidth flows.

B Single List vs Multiple Lists

The *multiple-list identification* scheme identifies flows that receive losses in K out of M drop-list intervals. This could be compared to the *single-list identification*, which would identify flows that receive the largest number of drops in a single, larger interval.

The main advantage of multiple-list identification over single-list identification is that multiple-list identification ignores flows that suffered drops only in one list. There are several reasons why a flow might have several drops in one list, but no drops in other 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 drops; or because of simple bad luck unrelated to the flow's sending rate.

In an environment with RED and a moderate packet drop rate, a flow is unlikely to receive multiple drops in a

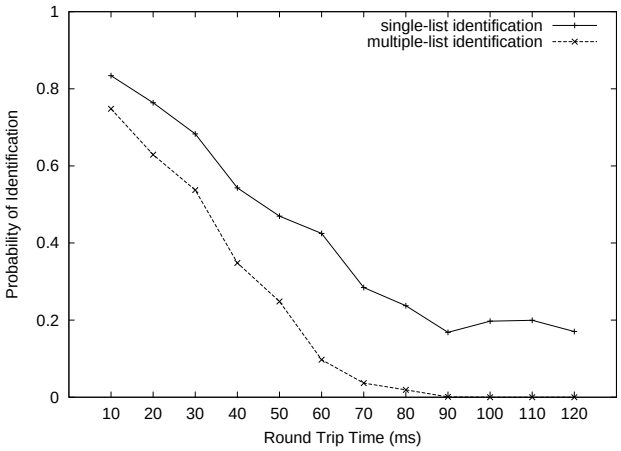

Figure 15: **The probability of identification for single and multiple list identification schemes for a bursty loss environment.** The target RTT was 40 ms.

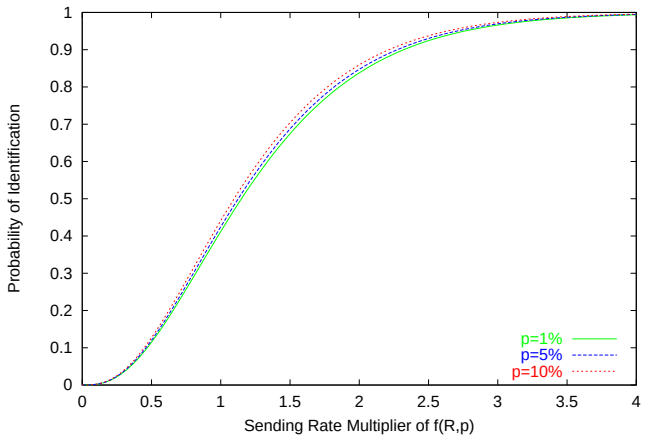

Figure 16: **The probability of identification of a flow sending at a rate $\gamma * f(R,p)$ for $K/M = 3/5$.**

single window of data, and therefore each loss event for a flow would be likely to consist of a single packet loss. In such an environment, there might be little difference between a single-list and a multiple-list identification scheme. However, in environments with higher drop rates or with a highly bursty arrival pattern, a multiple-list identification scheme could have significant advantages over a single-list identification scheme.

In a simulation we created an environment likely to show the advantages of a multiple-list scheme over the single-list scheme. The congested link had small buffer space, with the RED threshold min_{th} set to half of the buffer space. This scenario would be characterized by frequent buffer overflow, with multiple packets dropped from a window of data. Figure 15 shows the fraction of times a TCP flow with the given RTT was identified by the two schemes. The single-list scheme identifies a flow when it experiences K or more drops in the last $K * CL(R,p)$ seconds. It can be seen that in this environment a single-list scheme based on individual losses has a high probability of identifying conformant flows with round-trip times greater than R. In contrast, the multiple list scheme does a better job of identifying only the higher bandwidth flows.

C Probability of Identification for CBR flows

Section 5.1 showed the probability of identifying a TCP flow. The same is done here for a CBR flow. Consider a flow sending at $\gamma \times f(R,p)$ pkts/s, where p is the ambient drop rate and R is the target RTT. Since the length of the drop-list interval is given by Equation 3, the flow sends $\frac{\gamma K}{Mp}$ packets per drop-list interval. The probability $P(1)$ that the flow suffers at least one drop in a drop-list interval is

$$P(1) = 1 - (1-p)^{\frac{\gamma K}{Mp}}$$

For a flow to be identified, it has to suffer at least K drops in M drop-list intervals. So, the probability of this flow being identified is

$$\begin{aligned} P_{identification} = \\ & C(M,K)P(1)^K P'(1)^{M-K} + \\ & C(M,K+1)P(1)^{K+1}P'(1)^{M-K-1} + \\ & \cdots + C(M,M)P(1)^M \end{aligned} \quad (7)$$

$C(x,y)$ is the number of ways of choosing y out of x objects.

Figure 16 plots a flow's probability of identification as a function of its sending rate using the equation above. This result was confirmed using a controlled environment simulation where the drop rate at the queue was fixed. The graph should be interpreted carefully, x-axis

is the rate multiplier of $f(R,p)$, which itself is a function of the ambient drop rate p. That is, the sending rate multiplier γ of a CBR flow is different for different values of p. Therefore, as the ambient drop rate increases, a flow sending at a fixed rate in pkts/s becomes more likely to be identified.

D Response Time

In this section we do a simplified analysis of the time taken by RED-PD to control a high bandwidth CBR flow, as well as the time taken to release a flow which has reduced its sending rate.

D.1 Time to Cutdown

Assume that a flow increases its sending rate all of a sudden to $\gamma * f(R,p)$ pkts/s ($\gamma > 1$), where p is the prevalent drop rate at the output queue and R is the target RTT. We make the following simplifying assumptions in the analysis

1. The loss rate at the output queue is independent of this flow's arrival rate in the queue. In reality, the loss rate at the output queue can go up when the CBR flow suddenly starts sending at a high rate, and come back down again when the flow is controlled in the pre-filter.

2. This is the only flow whose dropping probability is being increased. From Equation (4), this means that the increase quanta of the dropping probability would be p.

3. The flow is successfully identified in each round. This is likely to be true until the flow is brought down to about twice $f(R,p)$. As seen in Figure 9 and 16, the probability of identifying the flow is high for a flow sending at twice $f(R,p)$.

The first assumption can lead to an overestimation of the response time if the increase in a flow's sending rate is responsible for an increased drop rate at the router, as would be typical in an environment with a low level of statistical multiplexing. The second assumption leads to an overestimation of the response time for flows with an increase quanta of more than p. This would be the case for flows sending at a very high rate, and hence the number of drops suffered by these flows would be more than the average among identified flows. The third assumption leads to a slight underestimation of the response time.

We calculate the time required to bring down the arrival rate of the flow in the output queue to $\alpha * f(R,p)$. The dropping probability required in the pre-filter in this case is $\frac{\gamma-\alpha}{\gamma}$. Because of Assumption 2, the dropping probability increase is in quantum of p. Hence the number of rounds required are $\frac{\gamma-\alpha}{\gamma p}$. Each round is $M-1$ intervals long because after increasing the pre-filter dropping probability, we wait for $M-1$ intervals and see if there are drops in K of last M drop-list intervals. This just speeds up the probability-increasing phase while maintaining the necessary time between two subsequent increments. Substituting the length of a drop-list interval from Equation 3, the total time required is:

$$t_{cutdown} = \frac{(\gamma-\alpha)RK(M-1)}{\gamma p\sqrt{1.5p}M} \tag{8}$$

D.2 Time to Release

We now estimate the time required to release a flow, that is, the time taken to transfer a monitored flow to the unmonitored category after it reduces its sending rate. The time estimate tells us not only how long this flow will be penalized after a rate reduction, but also the time required by RED-PD to forget a monitored flow which ceases to exist. The only assumption we make in this computation is that the flow is no longer identified after it cuts down its sending rate. The assumption holds as long as the reduced sending rate is much less than $f(R,p)$.

Consider a flow being monitored with a pre-filter dropping probability of P. This flow would become unmonitored when the pre-filter dropping probability goes below $P_{minThresh}$. In each round the probability is reduced by either a factor of β or a fixed amount p_d, whichever leads to lesser reduction (p_d is the upper bound on probability reduction in one step). Assume that there are n subtractive reduction rounds followed by m multiplicative reduction rounds.

The subtractive reduction rounds go in the series $P, P-p_d, ..., P-np_d$ and end when the dropping probability $P-np_d$ goes below $2*p_d$. Roughly, this gives us

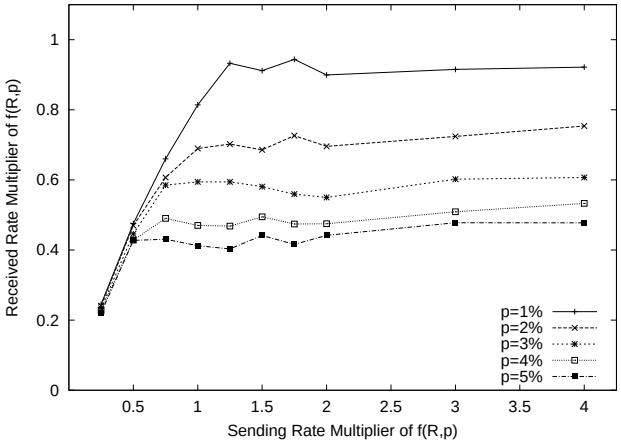

Figure 17: **Throughput of a CBR flow.**

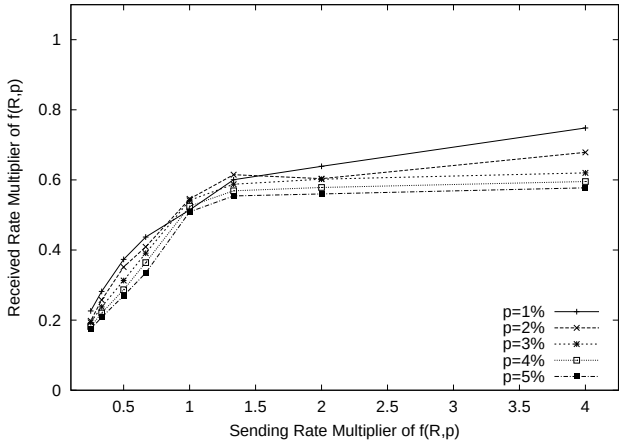

Figure 18: **Throughput of a TCP flow.**

$$n \geq \begin{cases} 0 & \text{if } P <= 2 * p_d \\ \frac{P}{p_d} - 2 & \text{otherwise} \end{cases}$$

The multiplicative reduction of the flow would go in the series $P - np_d, \frac{P-np_d}{\beta}, \frac{P-np_d}{\beta^2}, \cdots, \frac{P-np_d}{\beta^m}$, where $\frac{P-np_d}{\beta^m} \leq P_{minThresh}$. This gives us

$$m \geq \frac{log(\frac{P-np_d}{P_{minThresh}})}{log(\beta)}$$

The time required for each round is l drop-list intervals; this is the minimum wait between two successive decrements. Taking the drop-list interval length from Equation 3, the total release time is

$$t_{release} = \frac{(m+n)lRK}{sqrt(1.5p)M} \tag{9}$$

We use (β, p_d, $P_{minThresh}$, l) = (2, 0.05, 0.005, 3) for general monitored flows. For flows identified as unresponsive, we use (β, p_d, $P_{minThresh}$, l) = (1.5, 0.05, 0.0025, 5), which makes the release slower for unresponsive flows.

E Additional Simulations

This section contains more simulations done to evaluate RED-PD but were not included in the main body of the paper due to space considerations.

E.1 Persistent Congestion Throughput

The simulations in this section explore the extent of control RED-PD would wield on monitored flows in the event of persistent ambient drop rate at the queue, which can occur when the load presented by the unmonitored traffic is high. To simulate persistent ambient drop rate, the output queue is configured with an ambient drop rate. It drops each arriving packet with a fixed probability p (rather than as determined by the RED dynamics). Each simulation has a single source that is started with a sending rate of $\gamma \times f(R,p)$ pkts/s, where p is the configured drop rate at the output queue.

Figure 17 shows the throughput when the flow is sending CBR traffic as a function of its sending rate and the ambient drop rate. For a fixed ambient drop rate, there are several notable properties. The first is protection: very little bandwidth is shaved off flows that are sending much below $f(R,p)$). The second is effective control: a flow does not manage to gain extra bandwidth by sending more. The third property is fairness: a flow always gets at least as much bandwidth as obtained by another flow sending less than it.

The most striking feature in the graph is the increase in control (reduction in throughput) as the ambient drop rate increases. This result is a property of RED-PD's probability increase quantum, which is a function of the ambient drop rate. With a higher drop rate, the pre-filter dropping probability is higher. This means that at higher drop rates, RED-PD is ready to push down the monitored flows below $f(R,p)$ in an attempt to bring down

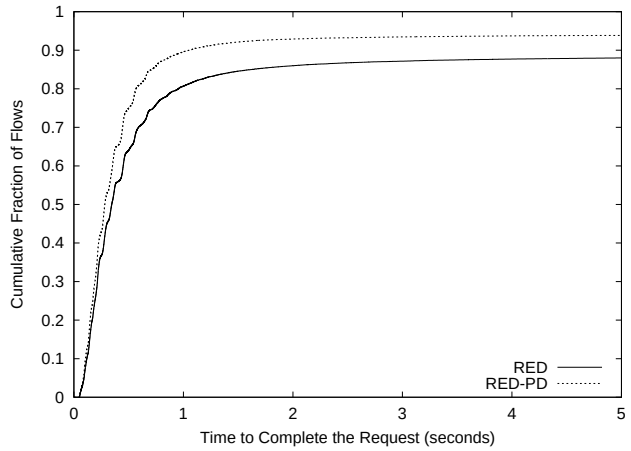

Figure 19: **Simulation with Web Traffic.** The cumulative fraction of requests which were completed before a given time

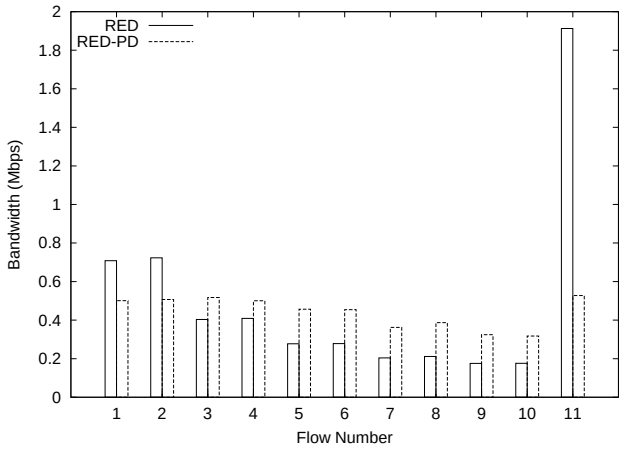

Figure 20: **Simulation with Web Traffic.** The bandwidth by the long flows. Flow numbers 1-10 are TCP flows, 2 each with RTT in ms of 20, 40, 60, 80 and 100. Flow 11 is a CBR flow sending at 2 Mbps.

the ambient drop rate at the queue. Moreover, it does so in a fair manner by also monitoring the flows sending below $f(R, p)$ but above the limit to which other monitored flows are reduced to.

A similar graph for TCP traffic is shown in Figure 18. It is interesting to note that, for a similar sending rate, while TCP flows get less bandwidth than CBR flows at low drop rates, they do better than CBR flows for higher drop rates. The differences are because of different probability of identification for TCP and CBR traffic.

E.2 Web Traffic

The simulations in Figure 19 shows the effectiveness of RED-PD in a dynamic environment in the presence of web traffic (as represented by the web traffic generator in *ns*) .

The object size distribution for the web traffic generator is Pareto with average 24 packets and shape parameter 1.2, and the packet size is 500 bytes. The long term average of the generated web traffic is about 5 Mbps, roughly 50% of the link bandwidth. A dumbbell topology with 5 nodes on each side was used. The RTTs for flows on this topology ranged from 20 to 100 ms, in 20 ms increments. In addition to the web traffic, traffic included one CBR flow with a sending rate of 2 Mbps and ten infinite demand TCP flows, 2 of each RTT.

Two simulations were run, one with and one without RED-PD. Figure 19 shows the cumulative fraction of web requests completed by a given time. The use of RED-PD results in more bandwidth available for the web traffic, in spite of the fact that short-RTT TCP flows carrying web traffic are also occasionally monitored (if they last sufficiently long to be identified). By monitoring the CBR flow and short-RTT TCP flows, RED-PD reduces the ambient drop rate, which does a lot of good for other traffic. Figure 20 shows the bandwidth obtained by each of the infinite-demand flows. Apart from the CBR flow, RED-PD also reduces the bandwidth obtained by the 20-ms TCP flows (1 and 2), as their RTT is less than the target R of 40ms.

E.3 Other Congestion Control Models

In this section we explore RED-PD's impact on congestion control models other than TCP. We use TFRC [FHPW00], a TCP-Friendly Rate Control mechanism that is less aggressive than TCP in its rate increases, and also responds to congestion more slowly. The broad conclusion from these simulations is that RED-PD functions well with both TCP and with TFRC, and does not adversely affect the relative fairness of TCP and TFRC.

TFRC is a rate-based protocol which attempts to smooth the sending rate while maintaining the same long term sending rate as TCP, as given by the TCP equation in

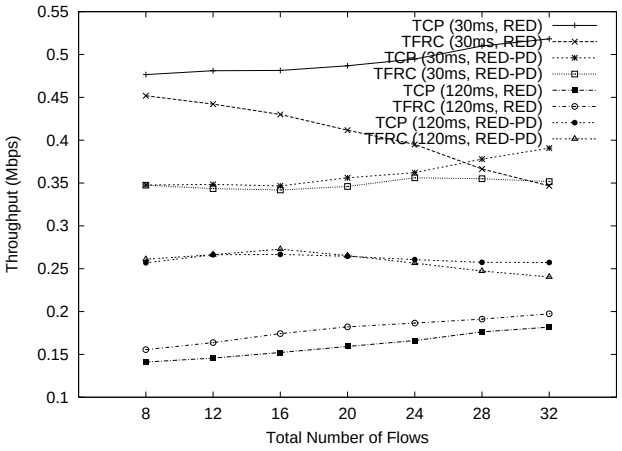

Figure 21: **The throughput of TCP and TFRC flows.** The graph plots the total throughput received by the four traffic types, both with and without RED-PD.

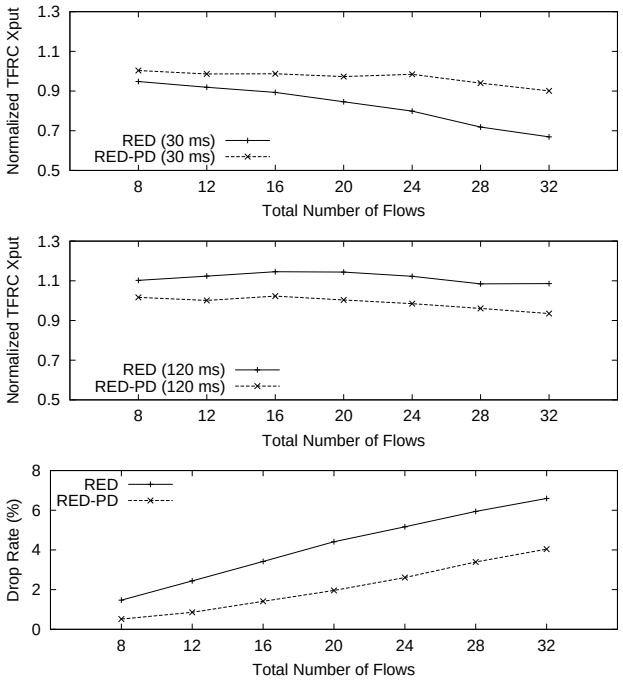

Figure 22: **TFRC performance compared with TCP.** The top graph plots the throughput of the 30-ms TFRC, normalized w.r.t. the throughput of the 30-ms TCP. The middle graph plots the same for 120-ms TFRC. The bottom graph plots the drop rate variation with number of flows.

[PFTK98]. Instead of halving its sending rate in response to each congestion indication, TFRC estimates the average loss rate, and adapts its sending rate accordingly. To maintain a smoother sending rate, TFRC responds more slowly to congestion than TCP. This raises the questions of interactions between RED-PD and TFRC, which we explore here.

For each simulation set we started $4 * n$ sources, for n ranging from 2 to 8. We used four traffic types, TCP and TFRC with an RTT of 30 ms, and TCP and TFRC with an RTT of 120 ms, and each simulation had n flows of each type. Each simulation set was run with and without RED-PD. Figure 21 shows the throughput received by four traffic types, averaged over five simulation sets, as a function of the number of flows ($4n$). Figure 21 shows that RED-PD significantly increases the bandwidth available for the 120-ms TCP and TFRC flows, while restricting the bandwidth available to the 30-ms flows. Figure 21 also shows that while with RED the 30-ms TFRC suffers during high congestion, relative to the 30-ms TCP, it is not the case with RED-PD.

Figure 22 gives the same data as in Figure 21, in a different aspect. It shows the average TFRC throughput normalized with respect to the average throughput of the TCP flow with the same RTT. It is evident that RED-PD is fairer than plain RED in its bandwidth allocation for both the monitored and unmonitored flows.

E.4 Multiple Congested Links

The simulations in this section explore the impact of RED-PD on flows traversing multiple congested links. Each congested link has a capacity of 10 Mbps. On each link eight TCP sources and two CBR sources were started, with round-trip times ranging from 20 to 80 ms. The CBR flows were sending at 4 Mbps each. We study the bandwidth obtained by a flow passing through all the congested links, as the number of congested links increase. The flow passing through multiple congested links is either a CBR flow with sending rate of 1 Mbps or (in a separate simulation) a TCP flow with an RTT of 80 ms. The RTT of the TCP flow was kept the same irrespective of the number of congested links it passed over by adjusting the delay of the connecting node, to factor out a throughput decrease due to an increasing RTT.

Figure 23 shows the bandwidth obtained by the flow passing through multiple congested links, with and without RED-PD. The throughput for the multiple-links

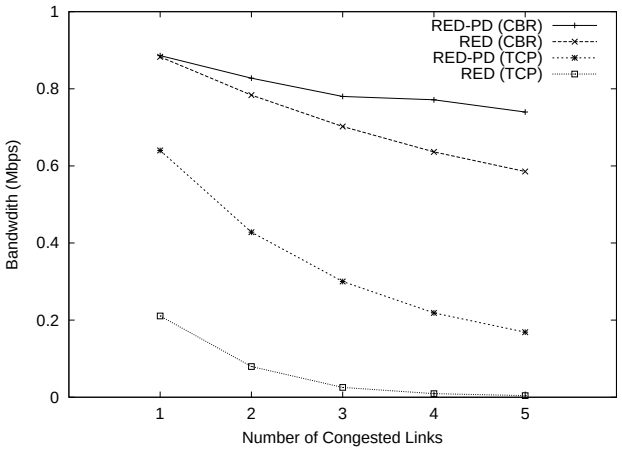

Figure 23: **Multiple Congested Links.** The graph shows the throughput of the CBR or the TCP flow which goes over all the congested links.

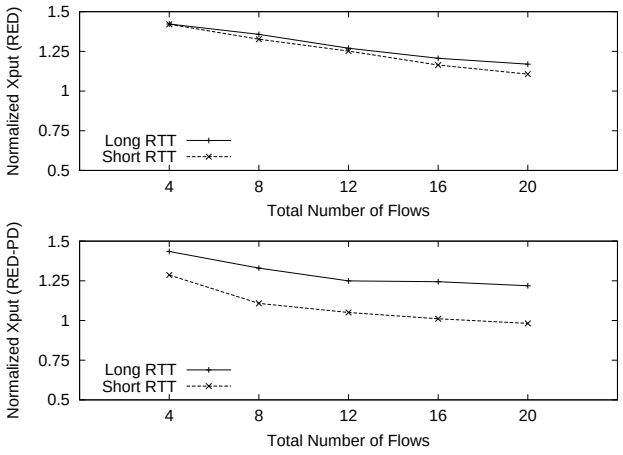

Figure 24: **RED-PD in byte mode.** The graphs show the throughput of the big packet (1000 bytes) TCP flows normalized w.r.t. the small packet (500 bytes) flows. The top graph shows the results with RED and the bottom with RED-PD.

flow goes down as the number of links increases, but is much better with RED-PD than with RED, because RED-PD decreases the ambient drop rate for each of the congested links. Unlike complete allocation schemes like FQ, RED-PD provides limited max-min fairness, and does not bring down the ambient drop rate to zero. However, by controlling the high-bandwidth flows on each link, RED-PD brings the ambient drop rate on it down to manageable levels, and thus, reduces the overall drop rate seen by flows traversing all of them.

E.5 Byte Mode

So far, we have not addressed the difference between the sending rate in bytes/s and pkts/s for flows with different size packets. There is no consensus in the networking community about whether fairness between flows should be in terms of packets or in terms of bytes, but a queue management system should be able to operate in both modes. Because RED-PD uses identification-based preferential dropping, the biases of RED-PD in terms of packet size are largely determined by the biases of the underlying Active Queue Management (AQM) mechanism.

RED-PD can be run in byte mode by running the underlying AQM mechanism in byte mode and counting drops in Equation 4 in terms of bytes. We tested RED-PD's performance in byte mode using simulation. The target RTT R was 40 ms and TCP flows with two different RTTs, 20 and 80 ms, were used. The two packet sizes used were 1000 bytes (big packet) and 500 bytes (small packet). In each simulation $4 * n$ flows were started, n each of a combination of RTT and packet size. The graphs in Figure 24 show the results with n ranging from 1 to 5. They plot the total throughput of big packet TCP flows normalized w.r.t to the small packet flows of the same RTT. The top graph, showing the results with RED, tells us that RED in byte mode has a bias towards large packet sizes. The bottom graph, with RED-PD results, shows that while RED-PD does not eliminate the bias completely it does help in reducing it significantly for monitored flows (short RTT flows).