Controlling High-Bandwidth Flows at the Congested Router

Ratul Mahajan and Sally Floyd[*]
ACIRI

November 20, 2000

Abstract

One weakness of the FIFO scheduling typical of routers in the current Internet is that there is no protection against misbehaving flows that send more than their share, or fail to use conformant end-to-end congestion control. This paper investigates RED-PD (RED with Preferential Dropping), a mechanism that uses the packet drop history at the router to detect high-bandwidth flows in times of congestion, and preferentially drops packets from these high-bandwidth flows to control the bandwidth received by these flows at the congested queue. This paper discusses the design decisions underlying RED-PD, and presents simulations evaluating RED-PD in a range of environments.

1 Introduction

The dominant congestion-control paradigm in the Internet is one of FIFO (First-In First-Out) scheduling at routers, in combination with end-to-end congestion control. This approach is simple to implement at the routers, and, because it involves no requirements for any uniformity of packet queuing, dropping, and scheduling algorithms in the routers along a path, it is well-suited to the heterogeneity and decentralized nature of the current Internet. At the same time, there are also serious weaknesses to such an approach. One such weakness is the inability to provide relative qualities of service for traffic traversing congested routers; this weakness is currently being addressed by the Differentiated Services Working Group in the IETF, and by other approaches as well.

A second weakness of a network based on FIFO scheduling is the vulnerability of the routers to end-nodes with nonconformant end-to-end congestion control. This is coupled with the intimate role played by a flow's round-trip time and packet size in the performance of the end-to-end congestion control, resulting in an effective allocation of bandwidth that is inversely proportion to the round-trip time of the flow. While there is no universally-agreed-upon fairness metric for

[*]This material is based upon work supported by AT&T. The authors can be reached at ratul@aciri.org and floyd@aciri.org, respectively.

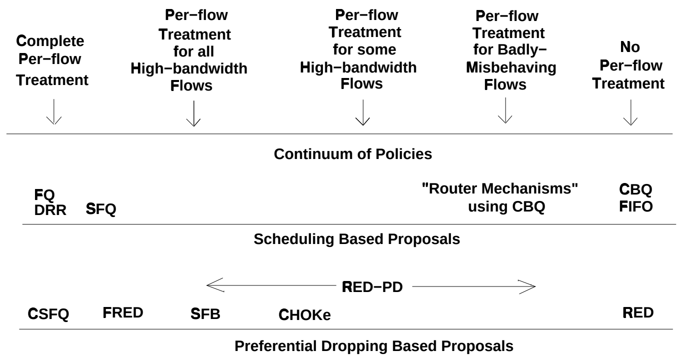
Complete Per−flow Treatment
Per−flow Treatment for all High−bandwidth Flows
Per−flow Treatment for some High−bandwidth Flows
Per−flow Treatment for Badly− Misbehaving Flows
No Per−flow Treatment
Continuum of Policies
FQ
DRR
SFQ
"Router Mechanisms" using CBQ
CBQ
FIFO
Scheduling Based Proposals
RED−PD
CSFQ
FRED
SFB
CHOKe
RED
Preferential Dropping Based Proposals

Figure 1: **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.

best-effort traffic in the current Internet[1], an unquestioning reliance on the end-to-end congestion control mechanisms implemented in the end nodes is shaky at best.

While all evidence is that the vast majority of the traffic in the current Internet uses conformant end-to-end congestion control (e.g., TCP), and while we know of no evidence of serious operational problems resulting from the current state of affairs, there is substantial agreement that additional mechanisms are needed at routers, at the very least to protect the Internet from "misbehaving" flows that don't use conformant end-to-end congestion control.

There is a continuum of possibilities in terms of per-flow treatment at the congested queue, shown in Figure 1, ranging from full per-flow scheduling mechanisms such as Fair Queuing on one end, to the complete absence of per-flow treatment typical of FIFO scheduling with active queue management such as RED on the other. The middle ranges of the continuum include mechanisms with per-flow treatment only for badly-misbehaving flows, mechanisms with per-flow treatment for all high-bandwidth flows, or mechanisms some-

[1]Max-min fairness is the leading contender for a fairness metric for competing best-effort traffic in the Internet, with proportional fairness as one of the competing metrics.

where in the middle with per-flow treatment for a small number of high-bandwidth flows. Approaches with limited per-flow treatment generally start with the identification of exceptional flows for special treatment, while approaches with full per-flow treatment are generally based on the direct allocation of bandwidth.

All of the proposed approaches work by applying some form of max-min fairness, restricting the bandwidth of a selected set of flows receiving the largest share of the bandwidth at the congested link. One advantage of max-min fairness is that it is easy to interpret locally, and makes no assumptions about behaviors elsewhere in the network. In applying a limited form of max-min fairness to a selected set of the high-bandwidth flows, it helps that bandwidth consumption in the Internet is highly skewed (see Section 3), with a small fraction of the flows on a link responsible for most of the bandwidth consumed over that link.

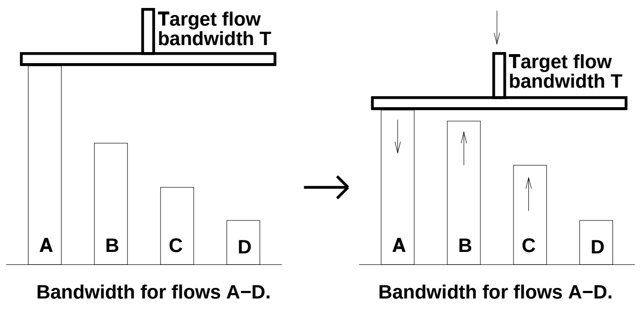

Figure 2: **Restricting flows to a target bandwidth T.**

In the simplest form, shown in Figure 2, this limited per-flow treatment can be thought of as selecting a certain target bandwidth T, and restricting the bandwidth of higher-bandwidth flows down to T. In the process, this increases the available bandwidth and decreases the packet drop rate for the aggregate of flows with arrival rates less than T. For example, when the bandwidth for Flow A in Figure 2 is restricted to T, this decreases the aggregate packet drop rate for the rest of the traffic, and allows Flows B, C, and D to receive increased bandwidth at the router, if those flows have sufficient demand to fill the available space. This is essentially the approach that we follow in this paper.

Mechanisms with per-flow treatment that apply to all high-bandwidth flows (for some definition of high-bandwidth) can be thought of as closely approximating full max-min fairness. In addition to applying a limited form of max-min fairness, several mechanisms propose actively punishing high-bandwidth flows judged to be violating end-to-end congestion control, as a positive incentive for the continued deployment of end-to-end congestion control.

1.1 Scheduling vs Preferential Dropping

The available mechanisms for per-flow treatment include both scheduling and preferential dropping. *Scheduling* approaches place flows in different scheduling partitions (there might be more than one flow in a partition) and the bandwidth received by each partition is proportional to that partition's scheduling rate. With *preferential dropping* mechanisms, different flows see different dropping rates, with the dropping rate as needed (in combination with end-to-end congestion control) to control the arrival rate of that flow. Scheduling mechanisms offer more precise control, while some preferential dropping mechanisms are fair only in the probabilistic sense. However, scheduling mechanisms also generally have higher requirements for per-flow state maintained at the queue. Figure 1 classifies the existing approaches as based on either scheduling or preferential dropping, and also roughly places them along the continuum mentioned above.

We note that preferential dropping mechanisms such as RED-PD or CSFQ have several advantages over scheduling-based schemes such as FQ. In particular, dropping-based schemes preserve FIFO scheduling, which is good for low-bandwidth flows with bursty arrival processes, where many per-flow scheduling mechanisms would introduce unnecessary delay for packets from such flows. Per-flow scheduling mechanisms could be amended to let pass small bursts from low-bandwidth flows, though this might introduce additional complexity to the scheduling mechanism.

Second, dropping-based schemes already incorporate active queue management to limit persistent queueing delay for packets within any flow; while per-flow scheduling mechanisms could also incorporate active queue management, there is little work outlining how this might be done.

Third, dropping-based schemes, if desired, can easily be used to actively punish high-bandwidth flows in times of congestion that are not using conformant end-to-end congestion control. Again, per-flow scheduling mechanisms could be amended to detect flows not using conformant end-to-end congestion control in a time of high congestion, and to give such flows less than their max-min fair share, as a concrete incentive to flows to use end-to-end congestion control. However, there is little work outlining how this would be done.

1.2 Overview of RED-PD

The approach in this paper, RED-PD (RED with Preferential Drop), is an identification-based approach using preferential dropping to control high-bandwidth flows. The use of preferential dropping enables us to use FIFO queuing which is advantageous because of its simplicity, ease of deployment and prevention of packet reordering (which can happen when an application has packets in two or more flows). The motivation of this work has been to develop a light-weight mechanism for identifying and preferentially-dropping from flows using significantly more than their 'share' of the bandwidth in a time of high congestion. We are particularly motivated by the need to control misbehaving flows not using conformant end-to-end congestion control. However, our mechanism includes a parameter that can be either adjusted, or set by the system administrator, to give behavior in a range of places in Figure 1's continuum of per-flow treatment.

Conceptually, there are two steps to an identification-based mechanism to control high-bandwidth flows. The first step is to identify high-bandwidth flows themselves and the second step is to reduce the bandwidth consumed by these flows. We say that the identified high-bandwidth flows are *monitored* flows.

Our identification mechanism is based on the drops seen by a flow at the router. The identification scheme can be tuned to identify flows above a bandwidth threshold that is a function of the packet drop rate at the congested queue. The bandwidth threshold can be used to operate RED-PD along different points in the continuum of per-flow treatment shown in Figure 1.

The preferential dropping mechanism is driven by the identification process. The dropping probability of a flow is increased or decreased in small quanta to control the arrival rate of the flow to the output queue. A flow's preferential dropping probability is increased each time the flow is identified in the output queue, and decreased when the flow's drops in the output queue are significantly below the threshold for identification.

In addition, we do not want to invoke preferential dropping when there is not sufficient demand from the rest of the traffic, and the preferential dropping would result in an underutilized link. To avoid an underutilized link, the router does not invoke preferential dropping if the average queue size is small (e.g., when the average queue size is less than RED's minimum threshold). This limits preferential dropping when there is insufficient demand from the unmonitored packets.

1.3 Organization

The organization of this paper is as follows. In the next section we discuss related work. Section 3 discusses some trace-based results which justify our belief that controlling just a few flows gives significant control over bandwidth distribution to a router. This is the main reason for effectiveness of an identification-based scheme. Sections 4 and 5 describe our identification and preferential dropping mechanism in detail. A discussion of some issues related to RED-PD is contained in Sections 6 and 8. In Section 7 we evaluate the scheme using simulation. Section 9 discusses the relationships between aggregate-based congestion control at the router, and flow-based congestion control such as RED-PD. Finally, we conclude in Section 10.

2 Related Work

In this section we briefly describe some existing proposals for achieving complete or limited fairness. There are two different aspects of scheduling mechanisms, namely fairness, and providing incentives for end-to-end congestion control [FF99]. We believe that end-to-end congestion control is one of the major reasons for the Internet's success and we should build mechanisms that not only provide fairness among flows but also provide incentives for end-to-end congestion control. Per-flow scheduling mechanisms have been designed to achieve fairness, but have in general not yet addressed the issue of incentives for the use of end-to-end congestion control.

In Fair Queuing (FQ) [DKS89], packets are sent in the order in which the router would have finished sending them if it could send each packet one bit at a time. Deficit Round Robin (DRR) [SV95] differs from FQ in its implementation, but achieves a similar effect. Both of these mechanisms offer an upper bound on extra delay introduced over a hypothetical fluid model scheme. Both of these mechanisms also use per-flow queueing, and therefore are somewhat expensive to implement.

Stochastic Fairness Queuing (SFQ) [McK90] uses hashing to map a flow to a queue, thus reducing the lookup cost for the source-destination pair in FQ. This simplification comes at the cost of maintaining many more queues than active flows (at least conceptually) to avoid collisions; SFQ is complicated to implement, with moving hash functions and avoiding packet reordering. SFQ does not provide complete fairness since multiple flows can be hashed into the same queue.

The work in the paper draws heavily from Core-Stateless Fair Queuing (CSFQ) [SSZ98] and Flow Random Early Detection (FRED) [LM97], 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. A key impediment to the deployment of CSFQ is that it would require an extra field in the header of every packet. Other drawbacks of CSFQ include the requirement that for full effectiveness, all the routers within the island need to be modified.

FRED is similar to CSFQ in that it uses FIFO scheduling, but instead of using information in packet headers, FRED constructs per-flow state at the router for those flows with packets currently in the queue. The dropping probability of a flow depends on the number of packets that flow has buffered at the router. FRED's fair allocation of buffers can yield very different fairness properties from a fair allocation of bandwidth [SSZ98]. In addition, the results obtained by FRED are not predictable, as they depend on the packet arrival times of the individual flows.

Stochastic Fair Blue (SFB) [FKSS99] does not use per-flow state to achieve fairer allocations but 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 only a few high-bandwidth flows. In the presence of multiple high-bandwidth flows it ends up punishing even the low bandwidth flows as more and more bins get polluted.

CHOKe [PPP00] is a recent proposal for approximating fair bandwidth allocation. 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 rationale behind this scheme is 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 only a few packets in the queue. The simulations in [PPP00] show that CHOKe achieves limited performance; for example, in the simulations the high-bandwidth UDP flows gets much more than their fair share.

Floyd and Fall in [FF97] briefly discuss mechanisms for identifying high-bandwidth flows using the RED [FJ93] drop history, using CBQ scheduling mechanisms to partition misbehaving and conformant flows in different classes. However, [FF97] did not present a complete solution, and the performance was limited by the choice of aggregate scheduling-based mechanisms instead of the per-flow preferential dropping mechanisms used in RED-PD. Our paper is in some sense a successor to [FF97] using the per-flow preferential-dropping mechanisms previously explored in FRED and CSFQ.

3 Why an Identification-Based Approach Works?

In this section we present a few trace results which justify the identification-based approach. The traces that we examined show the same results found by others, that a small fraction of flows are responsible for a large fraction of the bandwidth. We also show that identifying and preferentially dropping from these flows is useful. The results presented here are a subset of those presented in [MFP00] and are discussed here for the sake of completeness.

Figure 3 shows results from a one-hour-long trace taken from UCB DMZ in August 2000. The graph shows the fraction

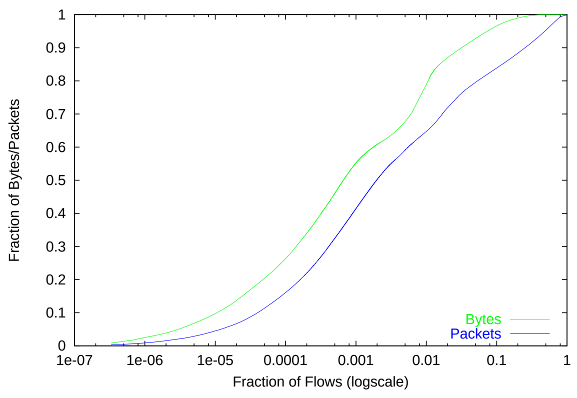

Figure 3: **Skewedness of bandwidth distribution**

of flows responsible for each 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. That means if a flow did not send packets for more than 64 seconds, it would be counted as a separate flow when it sends again. 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. Moreover, about 96% of the bytes and 84% of the packets came from just 10% of the flows. Though these numbers might seem very skewed, they are similar to those obtained from various header traces taken from NLANR [NLA], and to other results reporting on the heavy-tailed distribution of flow sizes. The numbers also don't change much if we change the timeout value. For instance, with a timeout of 2 seconds 1% of the flows got 78% and 57% of bytes and packets and 10% of the flows got 97% and 81% of the bytes.

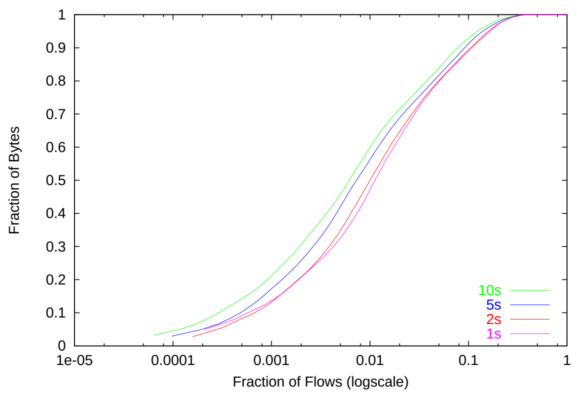

Figure 4: **Skewedness over smaller time scales**

The graph in figure 4 plots the same information for much shorter time windows. It shows the fraction of flows responsible for each 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 much shorter time windows. This is a useful piece of information for identification-based fairness approaches as they are likely to identify the high-bandwidth flows on fairly short time scales.

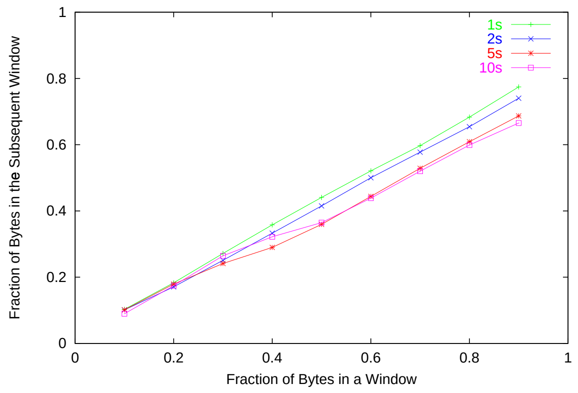

Figure 5: **Predictive nature of bandwidth consumption**

For an identification-based approach to be successful, it is necessary that the identified high-bandwidth flows in a given interval are a good predictor of the high-bandwidth flows in the succeeding interval. Figure 5 proves that this is indeed the case. The graph 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. So if an identification-based scheme was to restrict just these small fraction of flows, it could save a fair amount of bandwidth for other flows if it needed to.

In this paper we are particularly motivated by the need to control misbehaving flows at a router using cheap mechanisms. However, in the absence of reliable differentiation techniques between conformant flows with short round-trip times on the one hand, and high-bandwidth non-conformant flows on the other, our proposed scheme controls both categories of flows equally. For UDP flows such as non-congestion-controlled multimedia traffic, we would expect for the bandwidth consumed in one interval to be a plausible predictor of the bandwidth consumed in the subsequent interval, but this will not necessarily be the case for all misbehaving flows. There is a great need for more measurement data on the presence and behavior of misbehaving high-bandwidth flows in the Internet.

4 Identifying High-Bandwidth Flows

This section and the next describe RED-PD in detail. While this section talks about the identification mechanism, the next discusses the Preferential Dropping scheme.

We first list the possible techniques to identify misbehaving or high-bandwidth flows, and then delve into the approach taken by RED-PD. A router with no limitations in terms of memory or CPU cycles could identify high-bandwidth flows by calculating directly both the arrival rate and the packet loss rate for each flow over a given time interval. In this case, the router could use the direct measurement of the arrival rates to identify the high-bandwidth flows. Moreover, the router could identify misbehaving flows by using the measurements of drop rates and arrival rates, and plugging these into the TCP throughput equation.

However, keeping such a complete list of the arrival rate and the packet drop rate at the router for each flow is not necessary. In addition, real routers do have limitations in terms of memory or CPU cycles. We list below some of the range of possible techniques for identifying high-bandwidth flows without directly calculating the arrival rate for each flow. Possible techniques include a random sampling of the arriving traffic; using the history of packet drops as a somewhat-random sample of the arrival rate; and using other forms of history based on packet arrivals.

- **Random Sampling:** A router randomly samples incoming traffic. The high-bandwidth flows are then the ones with more samples.

- **Drop History:** Flows with high arrival rates are likely to have more packets dropped at the router. If a router goes back and observes its drop history, it can identify the high-bandwidth flows as being those with a large number of drops. [FF97] shows that the RED drop history can be successfully used to estimate the arrival rate of a flow. While not as precise as pure random sampling, the use of the drop history gives information about a flow's drop rate as well as a rough estimate of the flow's arrival rate.

- **History Data Structure:** A router could maintain some historical information, based on which it can estimate a flow's sending rate. The historical information itself is updated at each packet arrival. The zombie list in [OLW99] is an example of such an approach. More information about a flow in the history means that a flow is high-bandwidth.

- **Bloom Filters or Hashing:** Incoming flows could be hashed into bins at one or more levels, and the bins with more hits could be used to identify high-bandwidth flows. The identification mechanism in Stochastic Fair

Blue [FKSS99] is an example of this approach. A hashing-based approach avoids the memory and processing required for full per-flow state, with the risk of incorrectly identifying low-bandwidth flows mapped to the same bins as high-bandwidth flows.

RED-PD uses the RED drop history to identify high-bandwidth flows (though we use it in a way different from [FF97]). We note that by using the RED drop history, we are identifying not only high-bandwidth flows, but occasional lower-bandwidth flows that have been unlucky, in that their loss rate at the router is not an accurate predictor of their actual arrival rate. This is discussed in detail later in this section.

4.1 Choosing Identification Parameters

In this section we consider the identification scheme's criteria for identifying a high-bandwidth flow. We assume an environment dominated by flows using either TCP or TCP-compatible congestion control.

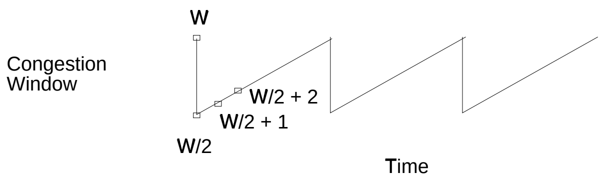

Figure 6: **TCP's congestion window in steady-state.**

Consider a queue with an average packet drop rate p over some recent interval of time. Next, consider a TCP flow with an RTT of r seconds. We consider a TCP flow where the receiver sends a separate acknowledgement for every data packet (rather than using delayed ACKs, and sending an acknowledgement for every two data packets). In the deterministic model of TCP with periodic packet drops, this TCP flow is doing its sawtooth with the congestion window varying between $w/2$ and w packets, with a packet dropped each time the window reaches w packets, as shown in Figure 6. A single congestion cycle includes roughly $w/2$ round-trip times, and lasts roughly $(w/2)r$ seconds. (For a TCP flow that only sends an acknowledgment for every two data packets, the length of a congestion cycle is somewhat longer.) In order to identify a high-bandwidth flow, we would ideally want to consider its arrival rate over several congestion epochs.

We let the average sending rate of a TCP flow with a round-trip time r and a steady-state packet drop rate p, in the deterministic model, be denoted as $f(r,p)$ pkts/sec. From some simple arithmetic [FF99], we have the following:

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

One could also use the equation $f_1(r,p)$ given in [PFTK98],

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

for TCP retransmit timeout value t_{RTO}, which can be approximated as $4r$. This equation $f_1(r,p)$ incorporates the effects of retransmit timeouts, and is based on a model of Reno TCP experiencing independent packet drops. We discuss the relative merits of $f(r,p)$ and $f_1(r,p)$, for our purposes, later in the paper. For the moment, we simply assume that our identification mechanism uses $f(r,p)$ instead of $f_1(r,p)$.

A congestion epoch contains $\frac{1}{p}$ packets, so the *congestion epoch length* CL is

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

seconds. Thus, given a steady-state packet drop rate p and average round-trip time r, we could consider the arrival rate of flows over several CLs.

Of course, there is not necessarily a "typical round-trip time" for the flows in a queue, and if there was, the router would not necessarily know what this typical round-trip time was. For our identification mechanism, we consider a *reference TCP* as a TCP connection, in the deterministic model, with a *target round-trip time* R and packet drop rate p. The *target* CL gives the CL for this reference TCP. Figure 7 shows how target CL varies with target RTT R and packet drop rate at the router. In an environment with a steady-state packet drop rate p, a flow sending at the rate of the reference TCP could receive a packet loss roughly once per CL.

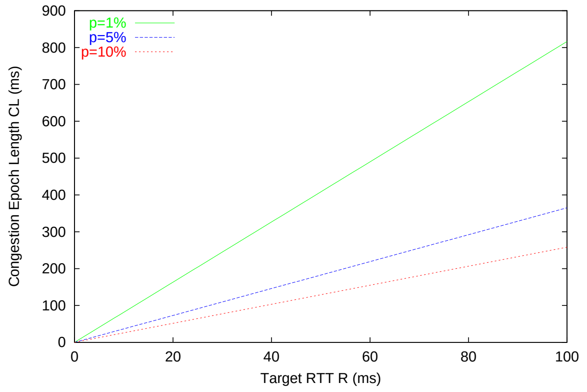

Figure 7: **Congestion epoch length CL for a target RTT R**

Our goal is to identify those high-bandwidth flows that, over a period of several (e.g., for $K = 3$) CLs, are sending at a rate higher than the reference TCP in the same environment. In addition, we restrict our attention to those high-bandwidth flows that have in fact gotten multiple congestion signals (losses) at the router during this interval. Packet loss samples have information about both the arrival rate and the loss rate of flows. 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 multiple loss events. We could instead identify flows by taking a random sample of the traffic unrelated to the current packet drops, but this would suffice only to estimate the arrival rate, and would not also confirm that the identified flow had actually received losses during that period.

Now we can reconsider our choice of $f(R, p)$ (from the deterministic TCP model) instead of $f_1(R, p)$ (from the Reno TCP model with timeouts) for defining the target CL. 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 CLs might be sending at a rate higher than $f_1(R, p)$ over that period. The equation for $f(R, p)$ is closer to the maximum sending rate of the TCP flow over the short term (of several congestion epochs). 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 CL is quite short, a flow could easily go for several CLs without receiving a retransmit timeout, and in this case it would seem important to use $f(R, p)$ to be properly conservative in our identification of high-bandwidth flows.

Guidelines for the choice of the target round-trip time R are discussed in detail in Sections 7.7 and 8.2. This parameter R is the single most important parameter for RED-PD's identification mechanism, and can be chosen to make RED-PD operate at different points along the per-flow continuum of Figure 1.

4.2 The Multiple-List Identification Scheme

RED-PD uses a *multiple-list identification mechanism* based on the packet drop histories over several successive *drop-list intervals*. For each drop-list interval (the interval length varies, see below), the router compiles a list of flows that have suffered drops in that interval. The high bandwidth flows are those which appear in these drop-lists "regularly".

For RED-PD, we consider the drop history of flows over $K = 3$ target congestion epoch lengths. In order to define the current target CL, we have to first have an estimate of the recent packet drop rate p (as measured over a period of roughly several CLs, for the most recent known value for CL). The router can reliably determine its loss rate as the number of drops divided by the number of arrivals. We use exponential averaging to smooth this drop rate estimate over successive intervals.

For RED-PD identification, we restrict our attention to flows that have received at least K separate loss events in the most recent K CLs. To do this, we divide the period of K CLs into M separate drop-list intervals, each of length

$$\frac{K}{M}\mathrm{CL} = \frac{\mathrm{K}}{\mathrm{M}}\frac{\mathrm{R}}{\sqrt{1.5\mathrm{p}}} \tag{3}$$

seconds, and keep a separate list of the dropped or marked packets in each interval. RED-PD *identifies* a flow if it has received drops in at least K of the last M drop-list intervals. Note that this implies that the length of a drop-list interval varies as a function of the recent average packet drop rate p.

Now, we consider our choice of the value for K, the number of CLs that make up our identification period. Larger values of K make the identification more reliable, but at the expense of an increase in the time required to identify high-bandwidth flows (see Section 5.2). In addition, a larger value of K makes 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. Smaller values of K make it more likely to catch unlucky flows, that is, low-bandwidth flows which happen to suffer drops. We use the absence of a drop in all the lists for decreasing a monitored flow's dropping probability (Section 5); a small value of K would lead to frequent changes in the dropping probability. For now, a value of $K = 3$ seems to give a reasonably prompt response along with a reasonable protection for unlucky flows that have received more than their share of losses. In particular, a value of $K = 2$ is considerably more vulnerable to the identification of unlucky flows (even the losses in the same window can spread over two lists).

After choosing K, the next parameter to consider is M, the number of separate drop-list intervals. Clearly we need M greater than K, because we want to only consider flows that have received at least K separate loss events. The minimum possible value of a drop-list interval to count the drops in the same window as a single event is the typical round trip time of flows in the queue. One would want each drop-list interval to be longer than this typical round-trip time to avoid over-counting number of separate loss events received by many flows. Given our choice of $K = 3$, a value of $M = 5$ has worked well for the number of separate drop-list intervals.

In any identification mechanism, it is important to strike a balance between the number of false positives and false negatives, and this desired balance also depends on the consequences of these false positives and false negatives. False positives occur for unlucky flows that receive more than their share of packet losses. If the consequences of a false positive is severe, then false positives should be avoided as much as possible. If desired, the router could protect itself against false positives by directly measuring the arrival rate of monitored flows, rather than relying solely on the packet loss rate as an indicator of the arrival rate. However, bad luck for a flow is a temporary phenomenon. In addition, the consequences of a flow being identified for the first time are not draconian, and further reduce its chances of being incorrectly identified again. Thus, we have not added extra mechanisms, such as the direct measurement of the arrival rate of monitored flows, to avoid false positives in our scheme.

False negatives in our identification scheme would result from flows that get lucky, in that they receive less than their share of packet losses. However, we rely on the underlying queue management mechanism to ensure that high-bandwidth flows do not get consistently lucky, that is, they do not consistently receive less than their share of packet drops. Thus, we can restrict our attention to flows that have actually received indications of congestion from this router.

4.3 Multiple Lists or a Single List?

The multiple-list identification scheme identifies a flow if the flow receives losses in K out of M drop-list intervals. This multiple-list identification could be compared to *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 several round-trip times with 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 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 identification scheme based on individual losses or on loss events, and there also 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 the single-list scheme based on loss events, which would itself have considerable advantages over a single-list identification scheme based on individual losses.

In a set of simulations we created an environment likely to show the advantages of a multiple-list scheme over the single-list scheme. The congested link had a small buffer space and a high min_{th} (half of the buffer space). Figure 8 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 seconds. It can be seen that in this environment a single-list

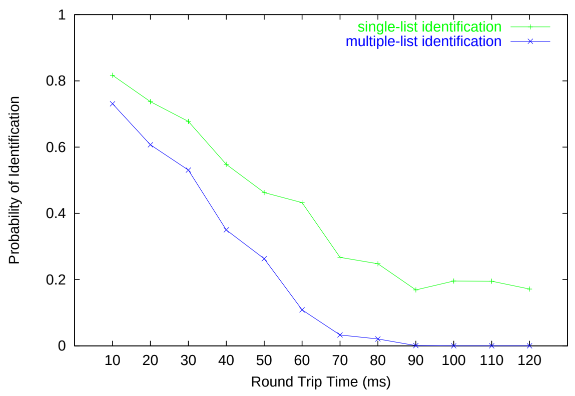

Figure 8: **The probability of identification for single and multiple list identification schemes for a bursty loss environment.** The target RTT was 40 ms.

scheme based on losses rather than loss events identifies flows with round-trip times much greater than R with a significant probability. On the other hand the multiple list scheme does a better job of identifying only the high bandwidth flows.

There is another possibility for a single-list scheme which would do better than the simple scheme above. This is a single-list scheme that identifies a flow if the flow receives K or more loss *events* in a single detection interval of duration $K * CL$, where a loss event is defined as one or more losses in a short period of time (such as a typical round-trip time). This scheme lends itself to a more complex implementation as timing information is required with every drop. We have not explored this loss-event-based single-list identification scheme; but we believe that it would be roughly equivalent to a multiple-list identification scheme with a drop-list interval equal to the time single-list scheme uses to filter drops into events.

4.4 Probability of Identification

In this section we compute RED-PD's probability of identifying a flow sending at a given fixed rate. We'll see that this probability is dependent not only on the flow's sending rate but also on the ambient drop rate. Note that in this section we are only investigating a flow's probability of being identified in a single round of the identification mechanism; the steady-state bandwidth received by a flow under RED-PD depends on whether or not the flow is persistently identified. The simulations in Section 7.1 show the bandwidth actually received under RED-PD by a flow sending at a fixed rate into a queue with a fixed ambient drop rate.

Consider a flow sending at a rate of $\gamma * f(R, p)$ pkts/sec, where p is the ambient drop rate and R is the target RTT chosen at the router. Assume that the length of the drop-list interval length is chosen according to the guidelines of the previous section. In this case, the flow is sending $\frac{\gamma K}{Mp}$ packets per interval. The probability $P(1)$ that a flow suffers at least one drop in a drop-list interval is as follows:

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

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

$$\begin{aligned} P_{identification} \;=\; & C(M,K)P(1)^K P'(1)^{M-K} + \\ & C(M,K+1)P(1)^{K+1}P'(1)^{M-K-1} + \\ & \cdots + C(M,M)P(1)^M \end{aligned} \tag{4}$$

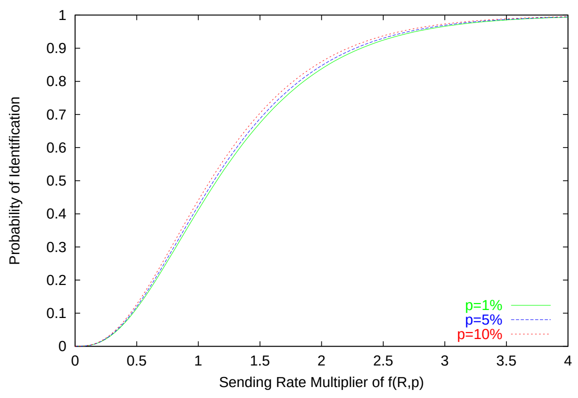

Figure 9: **The probability of identification of a flow sending at a rate $\gamma * f(R,p)$ for $K/M = 3/5$**

Figure 9 shows a flow's probability of identification as a function of its sending rate, as given the Equation 4. Separate lines show the probability of identification when the ambient drop rate is 1%, 5%, and 10% respectively. While the probability of identification approaches 1 for flows with higher sending rates, it is also non-zero for flows sending at less than $f(R,p)$ pkts/sec. Simulations of the identification of CBR flows, in a scenario where each packet is dropped independently with probability p, give essential the same probabilities of identification as those shown in Figure 9.

Figure 9 should be interpreted carefully, because the x-axis is the rate multiplier of $f(R,p)$, and $f(R,p)$ is itself a function of the ambient drop rate p. That is, a CBR flow sending at a fixed rate in pkts/sec might be sending at $2 * f(R,p)$ when the ambient drop rate is $p = 1\%$, but that same fixed sending rate will be $\gamma * f(R,p)$ for a much larger value of γ when the ambient drop rate is $p = 5\%$. Therefore, as the ambient drop rate increases, a flow sending at a fixed rate in pkts/sec becomes more likely to be identified.

Figure 9 shows that a flow can be identified even if it is sending at less than $f(R,p)$ pkts/sec. This occurs when the flow has been unlucky, and has received more than its share of 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. Also, the same flow is unlikely to be consistently unlucky in its packet drops as RED is not biased in any way towards a particular flow. These factors ensure a bandwidth distribution that is fair in average.

5 Preferentially Dropping High-Bandwidth Flows

After having identified high-bandwidth flows, we need to preferentially drop them to bring down the bandwidth consumed by them. We would like a differentiation mechanism to be light-weight, so that a number of flows could be monitored at the same time. We would like a differentiation mechanism to be compatible with FIFO scheduling, which is used by most routers in the Internet.

In addition, we would like a differentiation mechanism that not only protects other traffic from the monitored traffic, but also that provides relative fairness among the monitored flows, protecting monitored flows from other monitored flows. This rules out a solution that lumps all monitored flows together.

Finally, the differentiation mechanism should not only not starve the monitored flows, but also should not protect the monitored flows by giving them more bandwidth than they would have obtained in the absence of monitoring. This rules out solutions that only 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.

Before going into our preferential dropping mechanism, we discuss why existing packet-dropping mechanisms intended for aggregates are not suitable for our needs for per-flow differentiation. Two popular differentiation mechanisms are RIO [CF98] and WRED [Cis98]. In RIO, different average queue sizes are maintained for two categories of traffic, *in* and *out*. A RIO-like technique in our case would classify the identified high-bandwidth as *out* traffic and the rest as *in* traffic. A separate average queue size for the *in* traffic protects it from excessive *out* traffic. By putting together all monitored flows into the *out* category, RIO would fail to provide relative fairness between monitored flows. RIO also can lead to starvation of *out* traffic in some cases [BSP00].

WRED maintains only one average queue size and provides differentiation by having different drop probabilities (max_p in RED) and min and max thresholds associated with different categories of traffic. WRED fails to provide relative fairness between monitored flows if there is too much bad traffic and the average queue size exceeds min_th for the good traffic [BSP00]. WRED will also club together multiple monitored flows into the same class and thus fail to protect high-bandwidth flows from one another. It should be noted that the effectiveness of WRED goes down as more classes are introduced, so a different class for each monitored flow is not a good option. A large amount of good traffic will lead to starvation of monitored flows as the average queue size exceeds min_th of good traffic, which must be greater than or equal to min_th for the bad traffic [BSP00].

We now describe a technique, per-flow preferential dropping, which has all the required properties. The mechanism we use involves placing a pre-filter in front of the output queue (which uses FIFO scheduling). All the monitored high-bandwidth flows pass through this pre-filter, are dropped with a certain probability, and then are put in the output queue. Different monitored flows have different dropping probabilities. The unmonitored traffic would be put in the output queue directly. Figure 10 shows this process. This simple mechanism provides relative fairness between monitored flows; a high-bandwidth flow is dropped in proportion to its excess arrival rate, making its arrival rate in the output queue roughly the same as that of the highest-bandwidth unmonitored flow. Per-flow preferential dropping does not protect the monitored flows from the general congestion at the link, because the output queue does not differentiate between the flows in any manner once the pre-filter has cut down on the monitored traffic.

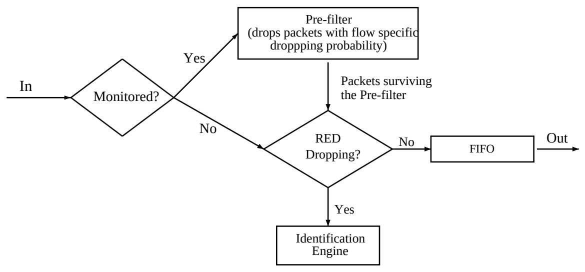

Figure 10: **The dropping mechanism.**

An important part of the above scheme is the dropping probability of each monitored flow in the pre-filter. We consider several possibilities for the pre-filter.

Token-Bucket-Based Preferential Dropping

In an ideal setting we would know the target bandwidth to which each monitored flow is to be restricted. One possibility for the dropping mechanism would be a token bucket or virtual queue restricting the monitored flow to the $target\_rate$ before its packets enter the output queue. A possible value for the $target\_rate$ would be the arrival rate given by the TCP response function for a flow with a default packet size, the round-trip time R used by the identification procedure, and the current steady-state packet drop rate at the output queue. In fact, this is the target rate implicitly used in our preferential dropping mechanism. While this would restrict the flow's arrival rate to the output queue to the desired value, it could lead to a somewhat bursty pattern of drops for the monitored flow. Also, the fast-path processing required for implementing a token bucket is more than just the probability lookup and drop required for the scheme we describe in Section 5.1.

Equation-Based Preferential Dropping

A second possibility for the dropping mechanism would be preferential dropping with drop probability $p = 1 - (target\_rate/arrival\_rate)$. This requires estimation of the flow's arrival rate at the router. The flow's arrival rate could be estimated from the packet drop history or measured directly, if there are only a small number of monitored flows. A direct measurement of the arrival rate of monitored flows would protect monitored flows that were simply unlucky in their packet drops. The arrival rate for a flow using end-to-end congestion control changes in response to packet drops and so does packet drop probability give by the equation above.

5.1 Identification-Based Preferential Dropping

A third possibility for the dropping mechanism, and the one explored in this paper, is preferential dropping driven by the identification mechanism. That is, if a monitored flow continues to be identified from its packet drops in the output queue, then 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, then its dropping probability in the pre-filter is decreased.

When a flow is identified as a flow to be monitored (as described in Section 4) for the first time,we start monitoring the flow, and packets from the flow are dropped in the pre-filter with a small initial dropping probability. We should point out here that the only drops considered by the identification process are those suffered at the entry to the output queue (shown by *RED Dropping* in Figure 10), not in the pre-filter. This means that the identification process is only 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. If the monitored flow is still identified by the identification process, it means that its arrival rate into the output queue is still on the high side. For such flows we increase the their dropping probability in the pre-filter. If the flow cuts down its sending rate and does not appear in any of y successive intervals in the identification process, we decrease its dropping probability in the pre-filter. Once the dropping probability of flow reaches a negligible value, the flow is unmonitored. With these small increases and decreases of the dropping probability, the router should settle around the right dropping probability for the flow.

The dropping probability for a monitored flow is not changed when the the flow appears in at least one but less than x of y successive drop lists. This provides the necessary hysteresis for stabilizing the dropping probability. Changes to the dropping probability are not made until a certain time period has elapsed after the last change. This ensures that the flow has had time to react to the last change.

Now we specify how the router increases and decreases the monitored flow's dropping probability. When decreasing the dropping probability, we just halve the dropping probability for a flow. The reduction is bounded by a maximum allowable decrease in one step. So if halving the dropping probability reduces the flow's dropping probability by more than this fixed bound, it is reduced by this bound instead. The upper bound on the reduction reduces oscillations. The absence of the flow in all the drop-lists could just be the result of it getting lucky and not from 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 dropping probability, two factors have to be considered, the drop rate in the output queue, and the arrival rate of the monitored flow. If the drop rate in the output queue is high, the arrival rate of monitored flows needs to be brought down sooner and hence the increase quanta should be high. Second, even among the monitored flows different flows have different sending rates, so the increase quanta should be different for different flows. That is, among the monitored flows, those flows with higher arrival rates to the output queue should receive higher drop rates in the pre-filter. (Flows with higher arrival rates to the output queue can be detected by their higher drop rates in the same queue.)

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. One 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 the following:

$$P_{delta_{flow}} = (drop_{flow} / avg\_drop\_count) * p \qquad (5)$$

where $drop_{flow}$ is the number of drops of this flow. If this increase quantum is more than the flow's existing drop rate, then 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. This dropping scheme has a fast reaction when drop rates are high, and differential treatment for flows based on their relative arrival rates.

5.2 Response Time

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

Assume that a flow increases its sending rate all of a sudden to $\gamma * f(R,p)$ pkts/sec ($\gamma > 1$), where p is the prevalent drop rate at the output queue and R is the target RTT. $f(R,p)$ is given by equation 1. 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 can go up when a flow starts sending at a high rate suddenly and come down as the dropping probability of this flow is increased in the pre-filter.
2. This is the only flow whose dropping probability is being increased. From equation 5, 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 of $f(R,p)$. As seen in Figure 9, the probability of identifying the flow is high for a flow sending at twice $f(R,p)$. The analysis can be made more accurate (and complex) by assuming a lesser probability and elongating the length in time required for each round, but we feel that is not necessary as we are mainly interested in a rough estimation.

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 (see Section 7.3). 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 only when we use the analysis below to calculate the response time to bring down the flow below $2 * f(R,p)$.

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 probability, we wait for $M-1$ intervals and see if there are drops in K lists out of last M. This just speeds up the probability increasing phase while maintaining the necessary time between two subsequent increments. Substituting the length of an interval from Equation 3, the total time required is

$$t_{cutdown} = \frac{(\gamma - \alpha)RK(M-1)}{\gamma p \sqrt{1.5p}M} \quad (6)$$

We did a simple simulation to test Equation 6. A CBR source was made to pass through a queue with a fixed configured loss rate. The CBR source initially sends at $0.25 * f(R,p)$ and increases its sending rate to $4 * f(R,p)$ at $t = 50s$. Figure 11 shows the results. The line marked “equation” is based on Equation 6, and the rest of the lines are simulation results for received bandwidth averaged over 1-second intervals. The target R used was 40ms. It is clear that the equation predicts the simulation results very closely till about $\alpha = 1.5$, below which Assumption 3 ceases to hold. The simulation line is slightly below the line of the equation because the simulation line plots the actual bandwidth received, while the equation plots the arrival rate into the output queue (after which the flow further suffers the configured loss rate).

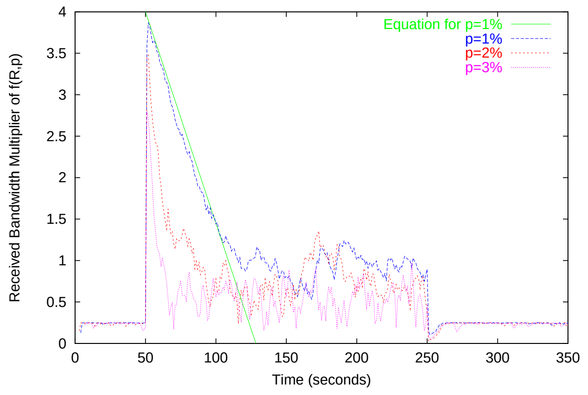

Figure 11: **The response time of RED-PD.** In this simulation the CBR source increases its sending rate to $4 * f(R,p)$ at $t = 50s$ and reduces it back to $0.25 * f(R,p)$ at $t = 250s$.

Figure 12 shows the importance of Equation 5 and proves that Assumption 2 leads to an overestimation of the time required to cut down the arrival rate of the flow. In this simulation two CBR flows were started. At $t = 50s$ one of them increases its sending rate to $4 * f(R,p)$, the other one to $2 * f(R,p)$. The line marked “2 flows” shows the bandwidth received by the former flow. The line marked “1 flow” is the same as that in Figure 11 and shown here for comparison. So when multiple flows are identified at the same time, a larger increase quantum for the higher senders leads to a quicker response from RED-PD and at the same time protects the low senders from a high increase quantum.

We now estimate the time required to release a flow, that is,

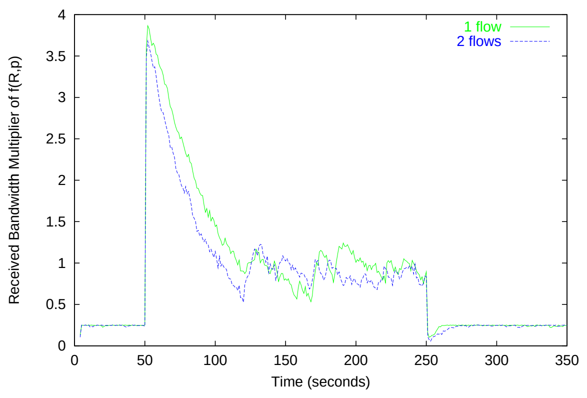

Figure 12: **Effect of Equation 5 and overestimation of response time because of Assumption 2.**

the time taken to transfer a monitored flow to the unmonitored category after it cuts down 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 way below $f(R,p)$.

Consider a flow being monitored with a pre-filter dropping probability of P. This flow would be unmonitored when this 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. Assume 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

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

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 intervals (minimum wait between two successive decrements). Taking interval length from Equation 3, the total release time is

$$t_{release} = \frac{(m+n)lRK}{sqrt(1.5p)M} \quad (7)$$

We use $(\beta, p_d, P_{minThresh}, l) = (2, 0.05, 0.005, 3)$ for responsive flows and $(\beta, p_d, P_{minThresh}, l) = (1.5, 0.05, 0.0025, 5)$ for unresponsive flows, which makes the release slower for unresponsive flows (see Section 6.2). For the simulation in Figure 11, $P = 0.75$ and the sending rate is $4 * f(R,p)$, so we get $n = 13$ and $m = 5$. The release time for $p = 1\%$ comes out to be 10.58 seconds and is very close to what we get from the simulations (immediately after the bandwidth cutdown at $t = 250s$).

6 Discussion

In this section we discuss some issues pertaining to the scheme we have described above.

6.1 Max-min Fairness

The fairness properties of the network are determined by the fairness properties of the routers and the congestion control algorithm used at the end hosts. In a network with FIFO queues and TCP, the relative bandwidth received by two conformant TCP flows depends on factors such as their round-trip times, and the number of congested links traversed by each flow. This section talks about the fairness properties of RED-PD and how it achieves a limited form of max-min fairness.

FIFO queues, without per-flow differentiation, cannot provide max-min fairness; the bandwidth received by a flow is proportional to the arrival rate of that flow. In contrast, a full max-min fairness scheme like FQ does not let a flow get more bandwidth than another flow whose demand has not been met. RED-PD aims to provide limited max-min fairness, in which we control the bandwidth allocated to the high-bandwidth flows. The high-bandwidth flows are defined using Equation (1) for a selected target RTT R and existing drop rate p. RED-PD changes the fairness properties of the system by controlling the arrival rate of selected flows into the output queue, by dropping from the identified flows in the pre-filer. Thus, the overall fairness would be similar to one in which no flow was sending at a rate greater than $f(R,p)$.

To put it clearly, in environments where all of the high-bandwidth flows are using conformant end-to-end congestion control, and have round-trip times considerably larger than the target round-trip time R, preferential dropping might never be invoked, leaving the fairness properties of the underlying system unchanged. That is, for such an environment with FIFO scheduling and TCP or TCP-compatible congestion control mechanisms, this would mean the familiar biases in favor of flows with smaller round-trip times.

However, in environments with non-conformant flows or with flows with round-trip times less than the target round-trip time R, RED-PD changes the bandwidth allocation of the underlying system. In particular, RED-PD preferentially drops flows till their arrival rate into the queue is not more than $f(R,p)$. Concomitant with the controlling the high bandwidth flows is the reduction of the *ambient* drop rate, defined as the drop rate at the output queue. The ambient drop rate is the drop rate seen by unmonitored flows, and also the drop rate seen by monitored flows after passing through the pre-filter. RED-PD reduces the ambient drop rate by controlling the arrival rate to the output queue. The extent of the reduction in the ambient drop rate q depends on the target RTT R used at the router (see section 7.7). The bandwidth above which flows are monitored also depends on the ambient drop rate. A decrease in the ambient drop rate results in an increase in the bandwidth allowed for an individual unmonitored flow. Extending this to the scenario where the drop rate is negligible, RED-PD has very little effect in such a scenario. This is the reason why we say RED-PD controls high-bandwidth flows at the congested router.

6.2 Unresponsive Flows

It is important for schemes that provide differential treatment for flows to provide incentives for end to end congestion control. The identification and preferential dropping mechanisms of RED-PD described so far make no judgments about whether an identified flow is or is not misbehaving - identified flows are treated the same, whether they are unresponsive, or simply TCP flows with short round-trip times. However, to provide a concrete incentive to end-users to use conformant end-to-end congestion control, one should actively *punish* high-bandwidth flows that are judged by the congested router as unresponsive. In this work, we have addressed the issue of identifying unresponsive flows only briefly, and at the moment RED-PD's only response to the identification of an unresponsive flow is to bring the flow down to its "fair share" somewhat more quickly that it would for a monitored flow that was not identified as unresponsive.

However, by performing the experiment of increasing the drop rate of an identified flow, the preferential dropping mechanism of RED-PD gives us the ideal framework for testing whether an identified flow is or is not responsive. Investigations 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. Some potential policies are dropping from unresponsive flows with a probability higher than that settled at by RED-PD dynamics or restricting unresponsive flows to bandwidth much lower than $f(R,p)$.

To perform the test for unresponsiveness, we keep a history of the arrival rate and drop rate for each monitored flow, where the drop rate is the sum of the pre-filter and output queue drops. Note that this would require the additional overhead of measuring the arrival rate for monitored flows. Our suggestion would be to record the arrival and drop rates over successive periods of yB seconds, which corresponds to several congestion epochs for any conformant TCP flows likely to identified.

Once we have a number of arrival/loss-rate pairs in a monitored flow's history, we can check if the packet drop rate for the flow has increased substantially while the flow was being monitored. If the flow's packet drop rate has increased substantially, but the flow's arrival rate at the router has not decreased in response, then the router can reasonably infer that the flow is not responsive. From Equation (1), the TCP response function $f(R, p)$ implies that if the long-term packet drop rate of a conformant flow increases by a factor x, then the arrival rate of the flow should decrease by roughly $\sqrt{x}$, that is, to roughly $1/\sqrt{x}$ of its previous value.

We would note that this test for unresponsiveness can have false positives, 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. Thus, the router should take some care in applying the results of the unresponsiveness test, and err of the side of caution.

The test for unresponsiveness can also have false negatives, in that it might not detect many high-bandwidth flows that are unresponsive. However, this is not a problem, since RED-PD controls the bandwidth received by these flows at any rate. The goal is simply that the most blatant and disruptive of the unresponsive flows would be identified as unresponsive.

To test for unresponsiveness, we simply test how a flow reduced its sending rate in response to an increased drop rate. If a monitored flow did not reduce its sending rate in response to an increased drop rate, we declare the flow as unresponsive. When a monitored flow is identified as unresponsive, the router is more wary of it than of other monitored flows. The router increases the dropping probability of the unresponsive flow in bigger quanta and decreases it in smaller quanta (the result of which can be seen in Section 7.3). The result is that we reach the right dropping probability for an unresponsive flow sooner, and keep it under tighter control. If the unresponsive flow becomes responsive, the flow will cut down its sending rate, and the router would slowly decrease the flow's dropping probability. When the dropping probability becomes negligible, we unmonitor the flow altogether and lose all history of the flow being unresponsive. Thus we do not have to do anything special to discover that an unresponsive flow has become responsive again.

6.3 Evasive Flows

Given a complete knowledge of the identification mechanisms at the router, a high-bandwidth flow can possibly evade the identification procedure. To evade the identification mechanism, the evasive flow would have to send in such a manner that it receives drops in at most $K - 1$ of M successive identification intervals. A flow is not likely to be able to do this precisely without knowing the length and start times of the identification intervals which are not fixed but change with 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, the identification mechanism could include additional procedures for detecting high-bandwidth but bursty flows. In particular, the identification mechanism could extend to flows that receive drops in at most $K - 1$ of M successive identification intervals, but that have a very high number of packet drops in these intervals.

6.4 Packets vs Bytes

So far, we have described RED-PD only in terms of packets, as a result of which a flow sending fewer larger packets can get away with more bandwidth than a flow with same bandwidth in bytes/sec but sending (more) smaller packets. Since there is no consensus in the networking community about whether flows should be charged per packet or per byte, one would desire a scheme which can be operated in either mode. The bias against smaller packets in RED-PD can easily be reduced by running the underlying RED queue in byte mode, where a flow's packets are marked in proportion to its arrival rate in bytes/sec, rather than in proportion to its arrival rate in packets/sec. For a RED queue running in byte mode, the counting of drops in Equation 5 should be done in terms of bytes instead of in terms of packets. With these changes, RED-PD's bias against smaller packets goes away. We tested this using simulations in which two groups of flows, with a packet size ratio of two, were sending at the same rate (in bytes/sec) over a common RED-PD queue with the above mentioned changes. One group of flows was sending at a rate more than $f(R, p)$ and the other at less than it (this is in some sense the worst case for the bias). In all the simulations we did, under different loss rate conditions, the bandwidth obtained by one group of flows was within 10% of another.

7 Evaluation

In this section, we determine, using simulation, the impact of RED-PD and its effectiveness in controlling high-bandwidth flows. Simulations were carried out using the NS network simulator [NS]. RED-PD has been added to the NS distribution, and we plan to make our simulation scripts available shortly. Most of the simulations in this section use TCP flows that send a separate ACK for every data packet; future simulations will also include TCP flows using delayed acknowledgements.

7.1 Received Bandwidth

The simulations in this section explore the bandwidth received by flows sending at a fixed rate, given RED-PD and a specified packet drop rate at the congested queue. The bandwidth reduction (by dropping) occurs in the pre-filter and output queue for monitored flows and just in the output queue for unmonitored flows. In order to have a controlled environment, the RED-PD output queue was configured to drop each arriving packet with a fixed probability p, rather than as determined by RED dynamics. A CBR source was started with a sending rate of $\gamma * f(R,p)$ pkts/sec $(\gamma > 0)$, where R is the target RTT (40 ms) and p is the configured drop rate at the queue. Figure 13 shows the results. The line labeled "precise" shows the target upper bound on the arrival rate of a monitored flow to the output queue, where the x-axis gives the steady-state arrival rate of the monitored flow to the pre-filter. The line labeled "p=1%" shows the actual throughput received by the CBR flow after the output queue, when the ambient packet drop rate is 1%. There are two reasons for the lines with the simulation results being below the "precise" target line. First, flows with an arrival rate of $f(R,p)$ pkts/sec to the output queue still have a non-zero probability of being identified, and of being preferentially dropped until they reduce their sending rate further. Second, the target line specifies the target arrival rate to the output queue, while the simulation line gives the throughput after the drops in both the pre-filter and the output queue.

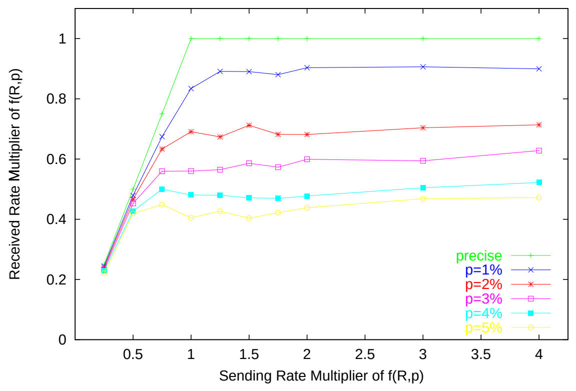

Figure 13: **Bandwidth received by a flow.**

The simulation lines in Figure 13 show that the received bandwidth of the CBR flow does not increase with the increase of the sending rate above $1 * f(R,p)$. This shows that a flow is not able to consume a large amount of bandwidth by increasing its sending rate. Another inference that can be drawn from the figure is that RED-PD successfully protects the low-bandwidth senders. This is evident by a relatively small decrease in bandwidth received by the flows whose sending rate is low. The only drops suffered by the low-bandwidth senders are those in the output queue. Thus, it becomes important to reduce the ambient drop rate by controlling the arrival rate of the high-bandwidth flows into the queue.

7.2 Fairness

The previous section showed that RED-PD can control the bandwidth allotted to high-bandwidth flows, and that the received bandwidth is roughly same for all the high-bandwidth senders. The last simulation, with just one flow passing over the queue with a fixed drop rate, was a "toy" simulation to prove the main properties of RED-PD. The rest of the simulations in the paper are normal simulations done with the underlying output queue as RED and hence, the ambient drop rate determined by RED dynamics. From this point onwards, we'll talk about a flow's throughput in terms of Mbps because there is no fixed drop rate p for calibrating the throughput in terms of $f(R,p)$. The capacity of the congested link is 10 Mbps in all the simulations. Flows were started at a random time within 10 seconds and aggregated results, when presented, were taken not before 20 seconds after the start of of the simulation. Unless otherwise specified, the target R is 40ms. Except for the web traffic simulation in Section 7.4, the packet size used in simulations is 1000 bytes. RED was running in packet mode and the SACK version of TCP was used in all simulations involving TCP.

The simulation in Figure 14 shows that RED-PD approximates max-min fairness among high-bandwidth flows (which are sending simultaneously and at different rates) by increasing and decreasing the drop probabilities in the iterative manner explained in Section 5. The simulation consists of 11 UDP CBR flows. The sending rate of the first flow is 0.1 Mbps, that of second flow is 0.5 Mbps, and every subsequent flow sends at a rate which is 0.5 Mbps more than its previous flow (so the last UDP flow sends at 5 Mbps). Figure 14 shows 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. The simulation shows RED-PD's ability to control the high-bandwidth flows, protect the low bandwidth flows (1 and 2), and protect high-bandwidth flows from each other. With RED-PD the ambient drop rate was reduced to roughly 4%, while without RED-PD the ambient drop rate would have been about 63%.

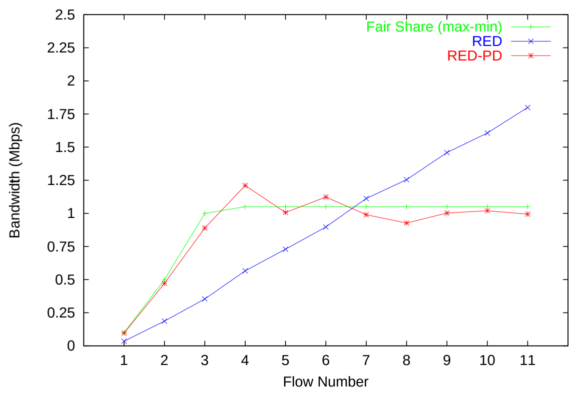

Figure 14: **Simulation with multiple UDP flows.** Flow 1 is sending at 0.1Mbps, flow 2 at 0.5 Mbps and every subsequent flow is sending at a rate 0.5 Mbps more than the previous flow

The simulation in Figure 15 consists of a mix of TCP and UDP flows. The aim is to study the effect of high-bandwidth UDP flows on conformant TCP flows. There are 9 TCP flows and 3 UDP 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 RTT of 70 ms. The UDP flows, with flow numbers 10, 11 and 12, have sending rates of 5 Mbps, 3 Mbps and 1 Mbps respectively. Again, Figure 15 shows the bandwidth of each of the 12 flows with RED and with RED-PD. With RED, the high-bandwidth UDP flows run away with almost all the bandwidth, leaving little for the TCP flows. In contrast, RED-PD is able to restrict the bandwidth allotted to the UDP flows to near their fair share. With a target R of 40 ms, RED-PD monitors not only the UDP flows but also those TCP flows with RTTs around 30 ms (and occasionally those with 50 ms as well). Clearly each of the UDP flows received a different pre-filter dropping rate, because each UDP flow was successfully restricted to its max-min fair share. It is interesting to note that the TCP flows with RTT of 70 ms still get slightly less bandwidth compared to other flows. We delve into the reasons for this in detail in Section 7.7.

7.3 Adapting dropping probability

The simulation in Figure 16 shows RED-PD's ability to adapt to the varying sending rate of a flow. The simulation is similar to that presented in Section 5.2 but it has a RED queue instead of a queue with a fixed drop rate, and is done in the presence of background traffic. A UDP flow was started with the initial rate of 0.25 Mbps. At $t = 50s$ the UDP flow increases its sending rate to 4 Mbps, and at $t = 250$ it decreases its sending rate back to 0.25 Mbps. Other traffic on the link consisted of 9 TCP flows with the same RTTs as above. Figure 16 shows

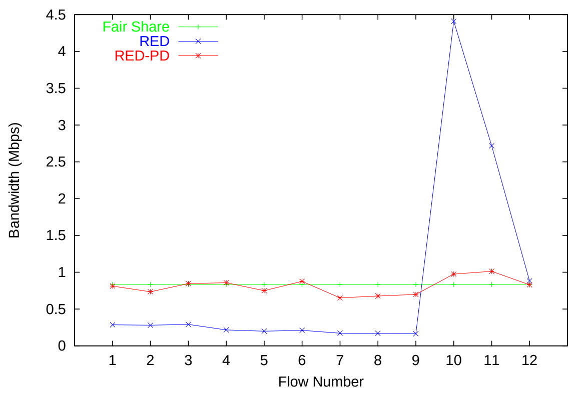

Figure 15: **Simulation with a mix of TCP and UDP flows.** Flows 1-9 are TCP flows, 1-3 with RTT 30ms, 4-6 with RTT 50ms and 7-9 with RTT of 70ms. Flow 10, 11 and 12 are UDP flows with sending rate of 5 Mbps, 3 Mbps and 1 Mbps respectively

the bandwidth received by the UDP flow averaged over 1-second intervals. Figure 16 shows the results of two separate simulations, one with the test for unresponsiveness disabled, and one with it enabled. The bottom graph of Figure 16 shows the exponential average (with $\alpha = 0.5$) of the ambient drop rate.

The response time for cutting down is much less than that computed in Section 5.2 mainly because Assumption 1 of a fixed ambient drop rate does not hold in this scenario, as shown in the bottom graph of Figure 16. The graph also shows that if the unresponsive test is *on*, the UDP flow is cut down much sooner and released later, as discussed before. In this simulation it took about three seconds for RED-PD to declare the UDP flow unresponsive after it increased its sending rate.

As we said earlier, the speed of RED-PD's reaction depends on the ambient drop rate and the arrival rate of the monitored flow. If the ambient drop rate is high, the increase quanta is large and the right dropping probability is reached much faster. In a similar fashion, RED-PD adapts its dropping probability when the conditions at the router change; for instance, if the arrival of a significant number of new flows leads to an increased ambient drop rate.

7.4 Web Traffic

The simulation in Figure 17 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 packet size in this simulation was 500 bytes. The object size distribution used was pareto with average 24 packets and shape parameter 1.2. The long term average of the generated web

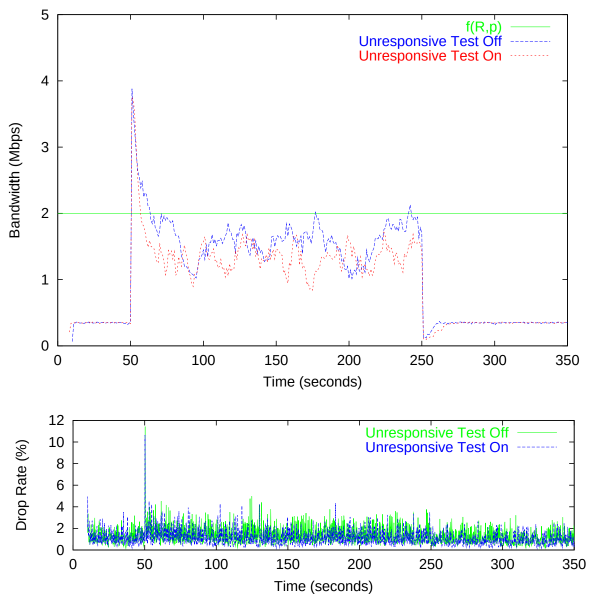

Figure 16: **Adapting Dropping Probability.** The top graph shows the bandwidth received by a UDP flow which changes its sending rate to 4 Mbps at $t = 50s$ and back to 0.25 Mbps at $t = 250s$. The line labeled $f(R,p)$ is based on the ambient drop rate seen over the whole simulation. The bottom graph plots the variation of ambient drop rate with time.

traffic was 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 addition to the web traffic, traffic included one UDP flow with a sending rate of 2 Mbps and ten infinite demand TCP flows. Two simulations were run, one with and one without RED-PD. Figure 17 shows the cumulative fraction of web requests completed by a given time. There is a significant gain for the web traffic with RED-PD, 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 UDP flow and short-RTT TCP flows, RED-PD reduces the ambient drop rate, which does a lot of good for other traffic. Figure 18 shows the bandwidth obtained by each of the infinite-demand flows. Apart from the UDP flow, RED-PD also reduces the bandwidth obtained by the short-RTT TCP flows (1 and 2), as their RTT is less than the target R of 40ms.

7.5 Multiple Congested Links

The simulation in Figure 19 explores the impact of RED-PD with multiple congested links. Each congested link has a capacity of 10 Mbps. On each link eight TCP sources and two UDP sources were started, with round-trip times ranging from

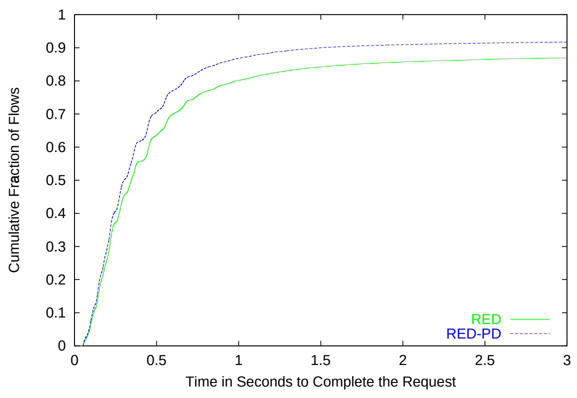

Figure 17: **Simulation with Web Traffic.** The cumulative fraction of requests which were completed before a given time

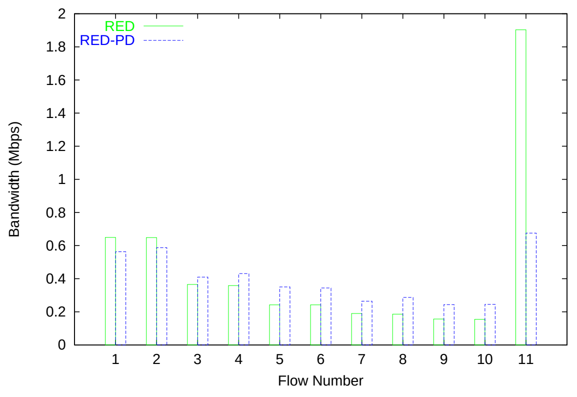

Figure 18: **Simulation with Web Traffic.** The bandwidth by the long flows. Flow numbers 1-10 are TCP flows, 2 each with RTT in ms of 20, 40, 60, 80 and 100. Flow 11 is a UDP flow sending at 2 Mbps.

20 to 80 ms. The UDP flows were each sending at 5 Mbps. 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 UDP 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 19 shows the bandwidth obtained by the flow passing through multiple congested links, with and without RED-PD. Each mark in Figure 19 shows the results of a single simulation, with either a UDP or a TCP flow, with or without RED-PD, and with the number of congested links ranging from one to five. The throughput for the multiple-links 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 has a goal of limited max-min fairness, and does not bring down the ambient drop rate to zero. However, by controlling the high-bandwidth flows, RED-PD brings the ambient drop rate down to manageable levels. The decrease in the ambient drop rate depends on the target R (higher R leads lower drop rate), an issue we discuss in more detail in Section 7.7.

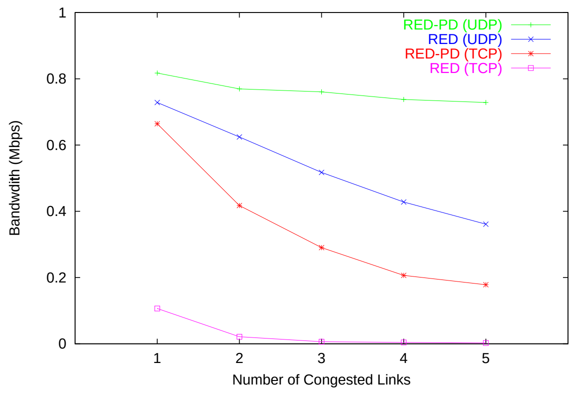

Figure 19: **Multiple Congested Links.** The graph shows the throughput of the UDP or the TCP flow which goes over all the congested links.

7.6 Other Congestion Control Models

In the simulation in Figure 21, we explore RED-PD's impact on congestion control models other than TCP. TFRC [FHPW00] is a TCP-friendly 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 [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 that does TCP. This can result in RED-PD penalizing a TFRC flow more than it would a corresponding TCP flow, when the levels of congestion at the router increase at a rate faster than the TFRC response time. However, we do not explore these transients in this work but restrict our attention to observing how the long term throughput of TFRC is affected with RED-PD instead of RED.

In the simulation we started $4 * n$ sources, n each of TCP and of TFRC with an RTT of 30 ms and n each of TCP and of TFRC with an RTT of 120 ms. The target R was 60 ms. Figure 20 shows the total throughput received by 30-ms and 120-ms TCP and TFRC sources, averaged over 5 simulations, as the number of flows increase. Figure 21 shows the TFRC

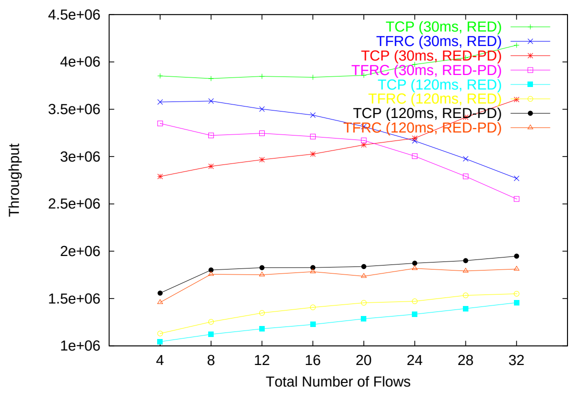

Figure 20: **The throughput of TCP and TFRC flows.** The graph plots the total thoughput received by that category (TCP or TFRC, RTT, RED or RED-PD) of flows

thoughput normalized w.r.t. to throughput of the TCP flow with the same RTT. In the top two graphs of Figure 21 each mark is from a separate simulation and the line joining them is the average. An increase in the number of sources is accompanied by a corresponding increase in the ambient drop rate as shown by the bottom graph.

It is evident that the performance of the short RTT TFRC flow deteriorates as the drop rate increases, with both RED and RED-PD. The relative TFRC throughput is better with RED-PD. Inspection of Figure 20 reveals that this improvement is largely because of the reduction in throughput of the 30-ms TCP flows with RED-PD. The drop in the absolute bandwidth obtained by the short-RTT TFRC flows is not significant (as they are already sending at a rate lower than the TCP of same RTT). There is a bandwidth gain seen by both long-RTT TFRC and TCP flows because of RED-PD. The different relative gains changes the bandwidth distribution from a mild bias towards TFRC (with RED) to a mild bias towards TCP (with RED-PD).

The broad conclusion from these simulations is that in general, RED-PD does not have an undesired impact on either TCP or TFRC in the presence of the other and the scheme is not vulnerable to TFRC, a TCP-friendly congestion control algorithm.

7.7 Simulations on Choosing R, the Target RTT

The simulations in this section illustrate how the choice of the configured round-trip time R affects the identification of flows for monitoring as well as the bandwidth received by monitored flows. Each column in Figure 22 represents a different simulation, with a different value for R, ranging from 10 ms to 170 ms. In each simulation 14 TCP connections

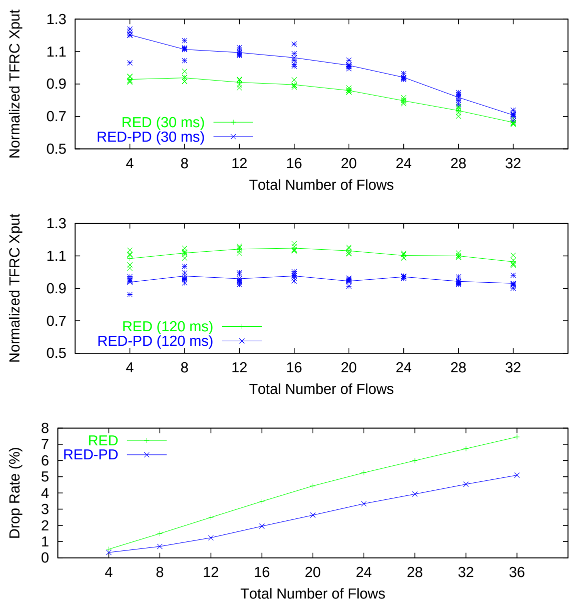

Figure 21: **TFRC performance compared with TCP.** The top graph plots the throughput of the 30-ms TFRC, normalized w.r.t. the throughput of the 30-ms TCP. The middle graph plots the same for 120-ms TFRC. The bottom graph plots the drop rate variation with number of flows.
fig:tfrc

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 22 shows the average bandwidth received by the TCP flows with round-trip times from 40-120 ms, while the bottom graph of Figure 22 shows the ambient drop rate. For the simulations with R less than 40 ms, RED-PD does not identify any flows, and the bandwidth distribution is essentially the same as it would be with plain RED. However, for the simulations with R of 40 ms or higher, the short TCP flows with a 40 ms RTT start to be identified and preferentially dropped. Note that as R is increased, the bandwidth received by the short TCP flows is decreased accordingly. This makes sense, because the target bandwidth for a monitored flow is $f(R, p)$, and this decreases with an increase of R. 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 22). Similarly, as R is increased above 80 ms, the 80 ms TCP flows begin to be monitored and preferentially dropped.

As these simulations illustrate, increasing RED-PD's configured value of R results in more and more flows being monitored, with more drops occuring in the pre-filter using per-flow state. 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 short flows.

The simulations in Figure 22 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 conformant flows with very short round-trip times.

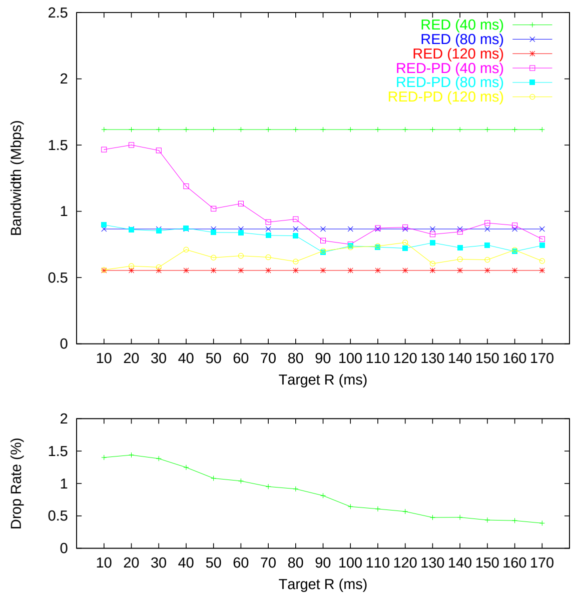

Figure 22: **The Effect of Target R.** The top graph shows the bandwidth received by 40 ms, 80 ms and 120 ms RTT TCP flows as R is increased. The bottom graph plots the ambient drop rate.

8 Other Issues with RED-PD

In this section we make explicit the state requirements of RED-PD, and also discuss the issues related to choosing the right target round-trip time R for a RED-PD queue.

8.1 State Requirements

In addition to the state needed by a regular RED queue, RED-PD requires state for the identification engine and for keeping track of monitored flows. For the identification engine, RED-PD stores M lists of flows which have suffered drops over an interval length given by Equation 3. The amount of memory required depends on both the ambient drop rate and the number of flows competing at that queue. The drop rate at the router is dependent on the configured value for R, which we discuss more in the next section. As an example, for an interval length of 150 ms, and $M = 5$, the router stored information about dropped packets over the past 750 ms. This should not be a problem even for high-speed routers. It should also be noted that this does not require fast memory, as the identification process does not run in the forwarding fast path. In rare cases when the router does not have enough memory to store headers for all the drops, it can 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 change in the behavior of RED-PD.

RED-PD also requires state to keep track of monitored flows. Upon the arrival of a packet at the router, RED-PD has to determine if this packet belongs to a monitored flow. If so, then RED-PD applies the appropriate preferential dropping to that packet before adding it (if it is not dropped) to the output queue. This preferential dropping occurs in the fast path and hence has to use memory that can keep up with the forwarding speed needed. Using a sparsely populated hash table (or perfect hash functions), fast lookups are possible. The speed required in this operation can be a deciding factor in how big a configured R a router can use. A larger R means that the router would be monitoring more flows, leading to larger hash tables and potentially slower lookups. While RED-PD could be configured to monitor many flows at one time, our own interest is in using RED-PD to monitor egregiously-misbehaving flows in times of high congestion. For this purpose, a fairly low value of R would be configured, and we would expect only a small number of flows to be monitored at one time. For this purpose, we do not expect that the state requirements for monitored flows to pose a problem.

8.2 Choosing R, the Target RTT

As was illustrated by the simulations in Section 7.7, 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, requires more state at the router, and comes closer to full max-min fairness. In contrast, a smaller value for R leads us to the opposite end of the spectrum. In addition, the desired choice of R depends on the likely mix of round-trip times for the conformant flows on the congested link.

Instead of a fixed, configured value for R, another possibility would be for R to be varied dynamically, as a function of the ambient drop rate and of the state available at the router. We plan to explore possibilities for dynamically varying R in later work.

9 Aggregate-based Congestion Control

RED-PD can be supplemented at the router by aggregate-based congestion control, where an aggregate might be traffic from a distributed denial-of-service attack, or a flash crowd of legitimate traffic to a web site related to a news-worthy event. With aggregate-based congestion control, when there is a rise in the packet drop rate, and RED-PD detects no individual flows responsible for this rise, then the router can check to see if the increased congestion is due to an increase in traffic from a traffic aggregate that is a subset of the traffic of the congested link. If the router is able to identify a traffic aggregate largely responsible for the traffic increase, the router might want to preferentially drop packets from that aggregate, to protect the rest of the traffic on that link from an overall increase in the packet drop rate. Coupled with this preferential dropping at the congested router, the router might invoke *Pushback* to request the immediate upstream routers to also drop packets from this aggregate. This use of Pushback prevents an unnecessary waste of bandwidth by packets that will only be dropped downstream. In the case of a denial-of-service attack, Pushback could help focus the preferential packet-dropping on the malicious traffic within the identified aggregate [BFM$^+$00].

At some level, aggregate-based congestion control can be thought of as a variant of RED-PD applied to aggregates rather than to individual flows, in that aggregate-based congestion control enforces an upper bound on the bandwidth given to an identified aggregate at the router in a time of congestion. However, there are substantial differences between flow-based (i.e., RED-PD) and aggregate-based (i.e., Pushback) congestion control at the router. As an example, the use of the TCP throughput equation is appropriate for individual flows (as defined by source and destination IP addresses and port numbers) but not for aggregates of flows. The distinction between conformant and non-conformant aggregates is considerably harder to pin down that that between conformant and non-conformant flows. For example, a conformant aggregate composed of many very short flows will response to preferential dropping differently than will an aggregate composed of a smaller number of large flows.

The identification mechanism is also somewhat different for flow-based and aggregate-based congestion control. An identification mechanism based on multiple lists of packet drops over successive time intervals is most appropriate for individual flows, where multiple packets within a round-trip time are defined as a single congestion event, but less appropriate for identifying a traffic aggregate; an identification scheme based on a single list of recent packet drops should be sufficient for identifying an aggregate. In addition, while the traffic at the router breaks down into a number of well-defined, mutually-exclusive flows, this is not necessarily the case with aggregates; a router's job could be to identify the aggregate responsible for high congestion, if there is one, from a large overlapping set of possible aggregates.

While Pushback is a key component for aggregate-based congestion control, it is not so crucial for flow-based congestion control schemes. For a flow-based congestion control mechanism such as RED-PD, when applied to a flow using conformant end-to-end congestion control, there is no need to pushback preferential dropping to an upstream router; increasing the packet drop rate at the congested router itself will be sufficient to reduce the arrival rate from that flow. When applying flow-based congestion control to a misbehaving flow, there could be some benefit in pushing back preferential dropping to an upstream router, but even so, the upstream routers for that flow are likely to be seeing all the packets from that flow, and could simply run RED-PD themselves and preferentially drop from that flow if it is causing significant congestion. In contrast, for aggregates, pushing back preferential dropping upstream can be a powerful tool, as aggregates change their composition from one router to the next. For example, at a router, the traffic aggregate defined by a certain IP address (source or destination) prefix could be composed of flows from a number of incoming links, so the traffic aggregate with that definition at a router one hop upstream could have a rather different composition.

10 Conclusions

In this paper we have presented RED-PD, a mechanism that uses the packet drop history at the router to detect high-bandwidth flows in times of congestion, and to preferentially drop packets from these high-bandwidth flows in order to control the bandwidth received by these flows at the congested queue. We showed the effectiveness of the proposed mechanism through extensive simulation, and we plan to run additional simulations in the future. In future work, we hope to explore an experimental deployment of RED-PD.

Acknowledgments

We are grateful to Scott Shenker for extensive discussions, and for reviewing an earlier version of this paper. Both of these have led to major improvements in the paper. We also thank Yin Zhang for discussions that resulted in the multiple list identification approach, Dina Katabi for helping with the simulation, Kevin Fall for his scripts from [FF97], and Mark Handley and Jitendra Padhye for TFRC. Ratul would like to acknowledge the support and guidance of David Wetherall and Tom Anderson.

References

[BFM$^+$00] Steve Bellovin, Sally Floyd, Ratul Mahajan, Vern Paxson, and Scott Shenker. Detecting and Controlling High Bandwidth Aggregates, 2000. Work in progress.

[BSP00] Ulf Bodin, Olov Schelén, and Stephen Pink. Load-tolerant Differentiation with Active Queue Management. *ACM SIGCOMM Computer Communication Review*, 30(4):pp. 4–16, July 2000.

[CF98] David Clark and Wenjia Fang. Explicit Allocation of Best-Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, Vol. 6(4):pp. 362–373, August 1998.

[Cis98] Cisco Web Pages: Distributed Weighted Random Early Detection. http://www.cisco.com/univercd/cc/td/doc/ product/software/ios111/cc111/wred.htm, February 1998.

[DKS89] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *ACM SIGCOMM*, September 1989.

[FF97] Sally Floyd and Kevin Fall. Router Mechanisms to Support End-to-End Congestion Control. Technical report, LBL, February 1997.

[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.

[McK90] Paul E. McKenney. Stochastic Fairness Queuing. In *IEEE INFOCOM*, June 1990.

[MFP00] Ratul Mahajan, Sally Floyd, and Vern Paxson. A Measurement Study of Traffic on a Wide-area Link, 2000. Work in progress.

[NLA] Nlanr web page: http://www.nlanr.net.

[NS] Ns web page: http://www.isi.edu/nsnam.

[OLW99] Teunis J. Ott, T. V. Lakshman, and Larry Wong. SRED: Stabilized RED. In *IEEE INFOCOM*, March 1999.

[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.

[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.

[SV95] M. Shreedhar and George Varghese. Efficient Fair Queuing using Deficit Round Robin. In *ACM SIGCOMM*, August 1995.