Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management

Sally Floyd, Ramakrishna Gummadi, and Scott Shenker
AT&T Center for Internet Research at ICSI

August 1, 2001, under submission

Abstract

The RED active queue management algorithm allows network operators to simultaneously achieve high throughput and low average delay. However, the resulting average queue length is quite sensitive to the level of congestion and to the RED parameter settings, and is therefore not predictable in advance. Delay being a major component of the quality of service delivered to their customers, network operators would naturally like to have a rough *a priori* estimate of the average delays in their congested routers; to achieve such predictable average delays with RED would require constant tuning of the parameters to adjust to current traffic conditions.

Our goal in this paper is to solve this problem with minimal changes to the overall RED algorithm. To do so, we revisit the Adaptive RED proposal of Feng *et al.* from 1997 [6, 7]. We make several algorithmic modifications to this proposal, while leaving the basic idea intact, and then evaluate its performance using simulation. We find that this revised version of Adaptive RED, which can be implemented as a simple extension within RED routers, removes the sensitivity to parameters that affect RED's performance and can reliably achieve a specified target average queue length in a wide variety of traffic scenarios. Based on extensive simulations, we believe that Adaptive RED is sufficiently robust for deployment in routers.

1 Introduction

End-to-end congestion control is widely used in the current Internet to prevent congestion collapse. However, because data traffic is inherently bursty, routers are provisioned with fairly large buffers to absorb this burstiness and maintain high link utilization. The downside of these large buffers is that if traditional drop-tail buffer management is used, there will be high queuing delays at congested routers. Thus, drop-tail buffer management forces network operators to choose between high utilization (requiring large buffers), or low delay (requiring small buffers).

The RED buffer management algorithm manages the queue in a more active manner by randomly dropping packets with increasing probability as the average queue size increases; the packet drop rate increases linearly from zero, when the average queue size is at the RED parameter minthresh (denoted by min_{th}), to a drop rate of max_p when the average queue size reaches maxthresh (denoted by max_{th}).[1] One of RED's main goals is to use this combination of queue length averaging (which accommodates bursty traffic) and early congestion notification (which reduces the average queue length) to simultaneously achieve low average queuing delay and high throughput. Simulation experiments and operational experience suggest that RED is quite successful in this regard.

However, as has been pointed out in [18, 19, 20] among other places, one of RED's main weaknesses is that the average queue size varies with the level of congestion and with the parameter settings. That is, when the link is lightly congested and/or max_p is high, the average queue size is near min_{th}; when the link is more heavily congested and/or max_p is low, the average queue size is closer to, or even above, max_{th}. As a result, the average queuing delay from RED is sensitive to the traffic load and to parameters, and is therefore not predictable in advance. Delay being a major component of the quality of service delivered to their customers, network operators would naturally like to have a rough *a priori* estimate of the average delays in their congested routers; to achieve such predictable average delays with RED would require constant tuning of RED's parameters to adjust to current traffic conditions.

A second, related weakness of RED is that the throughput is also sensitive to the traffic load and to RED parameters. In particular, RED often does not perform well when the average queue becomes larger than max_{th}, resulting in significantly decreased throughput and increased dropping

[1] In RED's gentle mode, which we employ here, the dropping rate increases linearly from max_p at an average queue size of max_{th} to 1 at an average queue size of $2 \times max_{th}$.

rates. Avoiding this regime would again require constant tuning of the RED parameters.

There have been several proposals for active queue management schemes intended to avoid these (and other) problems. We are currently preparing a detailed examination of the relative performance of these schemes. However, all of these schemes represent substantial departures from the basic RED design, and our purpose here is to look for a more minimal change to RED that can alleviate the problems of variable delay and parameter sensitivity mentioned above.

We think that the basic insight for such a solution lies in the original Adaptive RED proposal of Feng *et al.* from 1997 [6, 7]. This proposal retains RED's basic structure and merely adjusts the parameter max_p to keep the average queue size between min_{th} and max_{th}. In this paper, we describe a new implementation of Adaptive RED which incorporates several substantial algorithmic changes to the original Adaptive RED proposal while retaining its basic intuition and spirit. In addition, this new Adaptive RED algorithm automatically sets several other RED parameters; operators need only set the desired target average queue length.[2]

We have implemented this revised proposal in the NS simulator, and describe it here. This new version of Adaptive RED achieves the target average queue length in a wide variety of scenarios, without sacrificing the other benefits of RED. This not only leads to a more predictable average queuing delay, but also minimizes the possibility of "overshooting" max_{th}; thus, Adaptive RED reduces both the packet loss rate and the variance in queuing delay. Adaptive RED, thus, appears to solve the problem of setting RED parameters, which has been one of the banes of RED's existence.

While the Adaptive RED algorithm appears promising, we make no claims that this proposal is the only, or even the best, way to solve the problems addressed here. We present it as an existence proof that one need not abandon the basic RED design in order to stabilize the average queue length and "auto-tune" the other RED parameters. We also present Adaptive RED as a serious proposal for deployment in routers. Based on extensive simulations (only a subset of which we can report on here), and on the fact that Adaptive RED represents a small change to the currently implemented RED algorithm, we believe that that Adaptive RED is sufficiently robust for deployment.

This paper has eight sections, starting with Section 2 which discusses the metrics and simulation scenarios we use in evaluating Adaptive RED. Section 3 reviews some preliminary simulation results that illustrate RED's sensitivity to parameters and show that Adaptive RED does indeed address this problem. Section 4 describes the details of the Adaptive RED algorithm, including the automatic setting for the parameters maxthresh and w_q (the queue averaging constant). Simulation results of Adaptive RED in a variety of settings are presented in Section 5. Section 6 discusses the inherent tradeoffs between throughput and delay, and the difficult issue of determining the target average queue size. Related work is discussed in Section 7 and we conclude in Section 8.

[2] To avoid confusion and cumbersome terminology, in what follows the term original Adaptive RED will refer to the proposal of Feng *et al.* [6, 7] and the term Adaptive RED will refer to the revised proposal described here.

2 Metrics and Scenarios

In this section we describe the metrics and the range of scenarios that we used in evaluating Adaptive RED.

The primary goals of RED, or of active queue management in general, are to provide low average queuing delay and high throughput. Thus, for the evaluations of Adaptive RED in this paper we focus mainly on the metrics of average queuing delay and throughput. RED also has the secondary goals of improving somewhat upon the fairness given by Drop Tail queue management and of minimizing, for a given average queue length, the packet dropping or marking rate. We do not discuss the fairness behavior of Adaptive RED, since this is quite similar to the fairness behavior of RED. We only briefly consider the drop-rate behavior of RED and Adaptive RED, since degraded drop-rate behavior is generally reflected in degraded throughput.

A few comments about these metrics are in order. First, all of these metrics are router-based. While end-user metrics, such as file transfer times or per-packet delays, are important measures of an algorithm's validity, the end-user metrics of interest for Adaptive RED can be fairly easily derived from the router-based metrics we present, and we believe that the router-based metrics give more direct insight into the dynamics of AQM (Active Queue Management).

Second, we do not consider metrics related to the worst-case queuing delays because it is not a goal of AQM to control these worst-case delays. We envision AQM as being intended mainly for traditional best-effort traffic, where such worst-case guarantees cannot be provided with simple packet multiplexing in any case. To the extent that worst-case queuing delays are needed, they are controlled directly by configuring the queue's buffer size at the router (and thus are independent of the AQM algorithm).

Third, we do not consider metrics directly measuring queue length oscillations. While there has been substantial recent interest in queue length oscillations (see, for example, [8, 13]), we do not think that such oscillations are harmful unless they increase the average queuing delay or decrease the throughput, in which case the effects would be measured by our primary metrics. We discuss the impact of oscillations on our primary metrics in Section 5.1.

In evaluating Adaptive RED, we have explored a wide range of traffic scenarios, and have investigated the sensitivity of Adaptive RED to our simulation parameters. To verify robustness, we have considered performance for a range of workloads, levels of statistical multiplexing, and levels of congestion. Workloads include long-lived flows and short web mice, along with reverse-path traffic. The presence of data traffic on the reverse path introduces ack (acknowledgment) compression and the loss of ack packets, thereby increasing the burstiness of the data traffic on the forward path. Reverse-path traffic also forces a range of packet sizes on the forward path, as the forward path is now shared between data and ack packets. We also explore scenarios with changes in the workload or in the level of congestion over the course of the simulation. We have looked at dynamics with and without Explicit Congestion Notification (ECN). Finally, we have considered the effect of large window advertisements and different data packet sizes. We do not have space for all of these results in this paper, but a more complete description is available in [10].

3 The Motivation for Adaptive RED

Before delving into details of the design and performance of Adaptive RED, which we describe in Sections 4 and 5 respectively, we first review some simulation results illustrating RED's sensitivity to parameters, and showing that Adaptive RED does indeed address this problem. This section shows simulations illustrating RED's well-known characteristic of the average queue size and performance varying as a function of the RED parameters max_p and w_q. This section also includes simulation results with Adaptive RED, showing that by adapting max_p to keep the target queue size within a target range between min_{th} and max_{th}, it is possible to achieve the same performance of that from RED with a value of max_p tuned for that simulation scenario: in other words, Adaptive RED successfully "auto-tunes" the various RED parameters to achieve reliably good results.

Figures 1 through 3 show a set of simulations with a single congested link in a dumbbell topology, with the number of long-lived TCP flows ranging from 5 to 100. The long-lived flows have a range of round-trip times from 100 to 160ms, and the simulations include web traffic and reverse-path traffic. The congested link is 15Mbps.

The simulations in Figure 1 use RED with NS's default values of $w_q = 0.002$ and $max_p = 0.1$, and with min_{th} and max_{th} set to 20 and 80 packets respectively. RED is run in gentle mode in all of these simulations. Each cross shows the results from a single simulation, with the x-axis showing the average queuing delay in packets over the second half of the 100-second simulation, and the y-axis showing the link utilization over the second half of the simulation. Each line shows results from simulations with N flows, with lines for values of N ranging from 5 to 100. The crosses on each line show the results of simulations with max_p ranging from 0.5 on the left to 0.02 on the right. The packet drop rate in these simulations ranges from close to zero (for the simulations with five flows) up to 8% (for the simulations with 100 flows). As Figure 1 shows, the performance varies both with the number of flows and with max_p, with poorer throughput for those simulations with a larger number of long-lived flows. For these simulations, increasing the number of flows decreases link utilization, and increasing max_p leads to lower queue lengths. The downturn in utilization for low values of max_p reflects cases where the average queue length overshoots max_{th} a significant portion of the time.

As discussed later in Section 4.3, the queue weight of 0.002 is too large for a link bandwidth of 15Mbps, since it only averages the queue length over a fraction of a typical 100ms round-trip time. The simulations in Figure 2 differ from those in Figure 1 only in that they use w_q set to 0.00027 instead of 0.002. As is apparent from Figure 2, and discussed more thoroughly in Section 4.3, RED's performance is best when the average queue size is estimated over a small multiple of round-trip times, and not over a fraction of a single round-trip time. Figures 1 and 2 are evidence that RED's performance is sensitive to the value of the parameter w_q. Figure 2 also shows that non-adaptive RED can give quite good performance with this "good" value of w_q, but that the average queue size and throughput are both still a function of RED's parameter max_p. In particular, throughput suffers in the simulations when max_p is small relative to the steady-state packet drop rate, and the average queue size sometimes exceeds the max_{th} value of 80 packets. Thus, achieving good throughput and reasonable average queue lengths with RED requires careful tuning of both w_q and max_p. It is this careful tuning that Adaptive RED is designed to do automatically.

While we have yet to describe the Adaptive RED algorithm in detail (which we do in Section 4), the general design of Adaptive RED can be easily summarized as setting w_q automatically (based on the link speed) and adapting max_p in response to measured queue lengths. Later, in Section 5 we will explore the performance of Adaptive RED in more detail, but for now we show a few simulations that suggest that Adaptive RED does indeed remove the need to carefully tune these RED parameters.

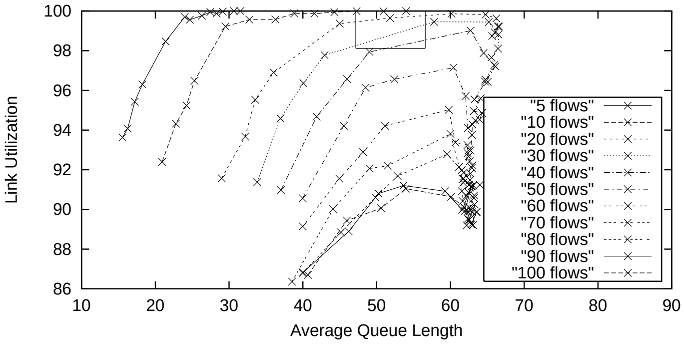

Figure 1: Delay-Utilization Tradeoff with RED, $w_q = 0.002$.

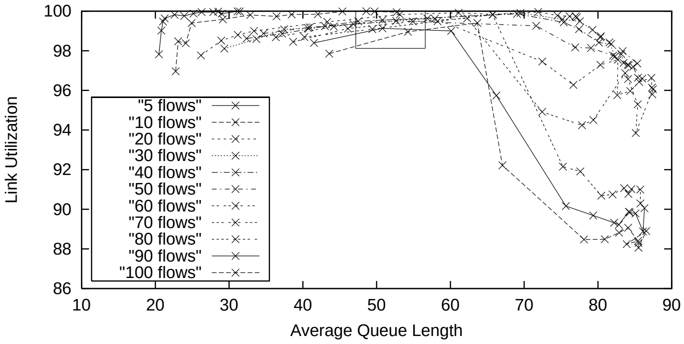

Figure 2: Delay-Utilization Tradeoff with RED, $w_q = 0.00026$.

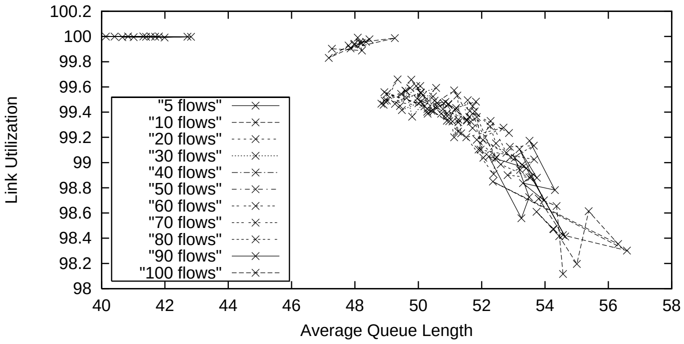

Figure 3: Delay-Utilization Tradeoff with Adaptive RED.

Figure 3 shows the same simulations with Adaptive RED; in this graph the various values of max_p are the initial values, and Adaptive RED adjusts them, as described in 4, in response to measured behavior. Note that the x- and y-axes of Figure 3 don't match those of Figures 1 and 2, so Figures 1 and 2 contain a box showing the area for Figure 3. The entire region depicted in Figure 3 occupies a small area in the "good" performance region of Figures 1 and 2. As in the earlier graphs, Figure 3 shows the results from the second half of a 100-second simulation; the clustering of the points for a given curve shows that the results are essentially independent of the initial value of max_p.

These simulations show that Adaptive RED, in setting w_q automatically and adjusting max_p in response to current conditions, is effective in achieving high throughput along with maintaining its average queue size within the target interval [44, 56]. This range corresponds to the algorithm's requirement of maintaining the average queue size within a pre-determined range around $(min_{th} + max_{th})/2$, as explained in Section 4. The only simulations with an average queue size outside that range are those with 5 flows; these simulations have few packet drops, a smaller average queue, and full link utilization.

The simulations with Adaptive RED all have high throughput, with the throughput ranging from 98% upwards (with 100 flows) to 100% (with 5 flows). For each number of flows, one could choose a static setting for max_p such that non-adaptive RED gives the same performance as Adaptive RED. The catch is that this static setting for max_p would have to be a function of the simulation scenario. For example, for the simulations with 20 flows, the performance of Adaptive RED corresponds roughly to the performance of non-adaptive RED with max_p set to 0.07, while for the simulations with 100 flows, the performance of adaptive RED corresponds roughly to the performance of non-adaptive RED with max_p set to 0.2.

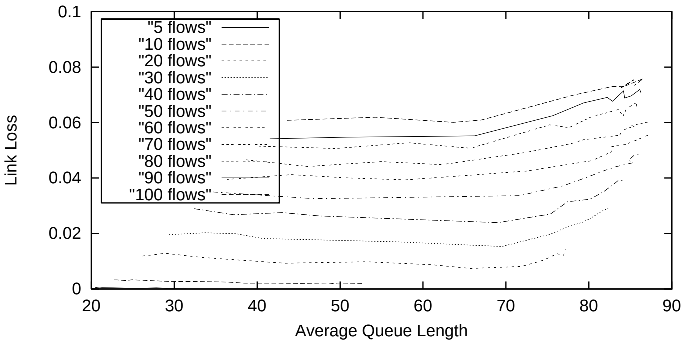

Figure 4: Delay-Loss Tradeoff with Normal RED, $w_q = 0.00026$.

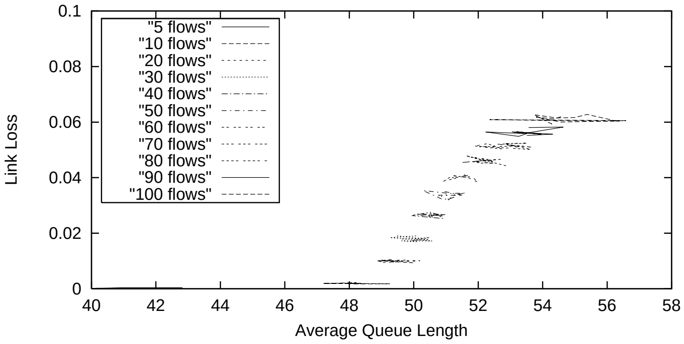

Figure 5: Delay-Loss Tradeoff with Adaptive RED.

Figures 4 and 5 show the packet drop rates from the simulations in Figures 2 and 3. They show that while both RED and Adaptive RED have roughly the same packet drop rates for a particular set of flows, Adaptive RED, by keeping the average queue size away from max_{th}, avoids the higher packet loss that RED incurs when the average queue size is around max_{th}. We have also explored fairness, and have verified that the fairness properties are similar in the simulations with RED and with Adaptive RED.

We have explored these simulations for a range of scenarios, including a range of link bandwidths and mixes of web traffic, with and without ECN, with the queue measured both in units of packets and in units of bytes, and with RED in byte mode (taking into account the size of a packet in bytes in deciding whether or not to drop the packet) and in packet mode. In all of these simulations, we see the same good performance from Adaptive RED.

3.1 Illustrating RED's Varying Queue Size

The previous simulations showed the steady-state performance of RED and Adaptive RED. We now investigate how RED and Adaptive RED respond to a rapid change in the congestion level. The simulations presented here illustrate RED's well-understood dynamic of the average queue size varying with the congestion level, resulting from RED's fixed mapping from the average queue size to the packet dropping probability. For Adaptive RED, these simulations focus on the transition period from one level of congestion to another.

These simulations use a simple dumbbell topology with a congested link of 1.5Mbps. The buffer accommodates 35 packets, which, for 1500-byte packets, corresponds to a queuing delay of 0.28 seconds. In all of the simulations, w_q is set to 0.0027, min_{th} is set to five packets, and max_{th} is set to 15 packets.[3]

For the simulation in Figure 6, the forward traffic consists of two long-lived TCP flows, and the reverse traffic consists of one long-lived TCP flow. At time 25 twenty new flows start, one every 0.1 seconds, each with a maximum window of twenty packets. This is not intended to model a realistic load, but simply to illustrate the effect of a sharp change in the congestion level. The graph in Figure 6 illustrates non-adaptive RED, with the average queue size changing as a function of the packet drop rate. The dark line shows the average queue size as estimated by RED, and the dotted line shows the instantaneous queue. The packet drop rate changes from 1% over the first half of the simulation, to 12.6% over the second half, with corresponding changes in the average queue size.

The graph in Figure 7 shows the same simulation using Adaptive RED. Adaptive RED shows a similar sharp change in the average queue size at time 25. However, after roughly ten seconds, Adaptive RED has brought the average queue size back down to the target range, between 9 and 11 packets. The simulations with Adaptive RED have a slightly higher throughput than those with non-adaptive RED, (95.1% instead of 93.1%), a slightly lower overall average queue size (11.5 packets instead of 13.4), and a smaller packet drop rate. The simulations with Adaptive RED illustrate that it is possible, by adapting max_p, to control the relationship between the average queue size and the packet dropping probability, and, thus, maintain a steady average queue size in the presence of traffic dynamics.

Figure 8 shows a related simulation with twenty new flows starting at time 0, and ending at time 25. The simulation with non-adaptive RED in Figure 8 shows the decrease in the average queue size as the level of congestion changes at time 25. This time, the packet drop rates with non-adaptive RED are 9.7% over the first half of the simulation, and .8% over the second half.

There is a similar change in the average queue size in the simulation with Adaptive RED in Figure 9, but within ten seconds Adaptive RED has brought the average queue size back to the target range. The simulation with Adaptive RED has a similar throughput to that with non-adaptive RED, (93% instead of 92.7%), and a slightly lower overall average queue size (11.1 packets instead of 12.4).

4 The Adaptive RED Algorithms

The overall guidelines for Adaptive RED as implemented here are the same as those for the original Adaptive RED from [6], that is, of adapting max_p to keep the average queue size between min_{th} and max_{th}. Our approach differs from original Adaptive RED in four ways:

- max_p is adapted not just to keep the average queue size between min_{th} and max_{th}, but to keep the average queue size within a target range half way between min_{th} and max_{th}.
- max_p is adapted slowly, over time scales greater than a typical round-trip time, and in small steps.
- max_p is constrained to remain within the range [0.01, 0.5] (or equivalently, [1%, 50%]).
- Instead of multiplicatively increasing and decreasing max_p, we use an additive-increase multiplicative-decrease (AIMD) policy.

The algorithm for Adaptive RED is given in Figure 10.

The guideline of adapting max_p slowly and infrequently allows the dynamics of RED—of adapting the packet-dropping probability in response to changes in the average

[3]The values for w_q and max_{th} are obtained from the guidelines in Section 4.3, while the value for min_{th} is a policy choice.

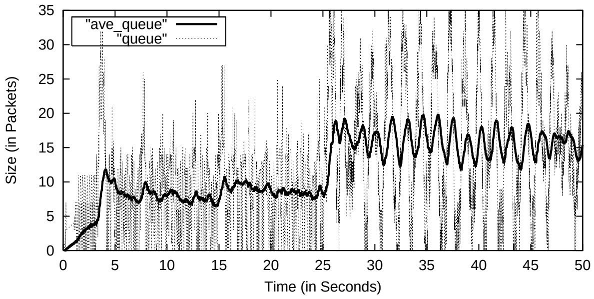

Figure 6: RED with an Increase in Congestion.

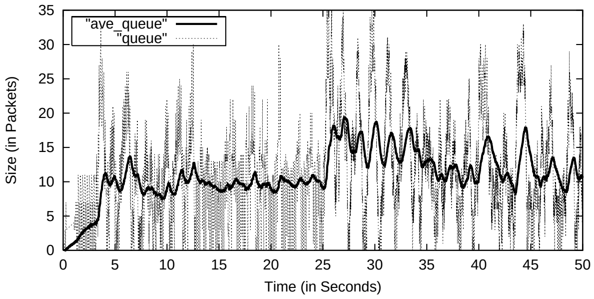

Figure 7: Adaptive RED with an Increase in Congestion.

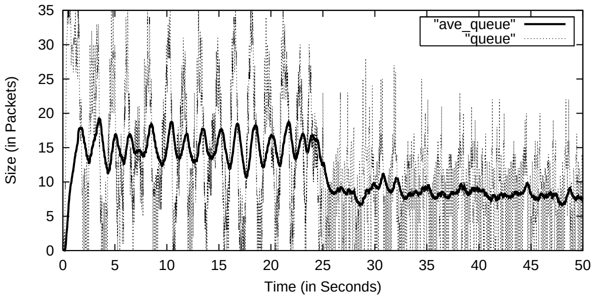

Figure 8: RED with a Decrease in Congestion.

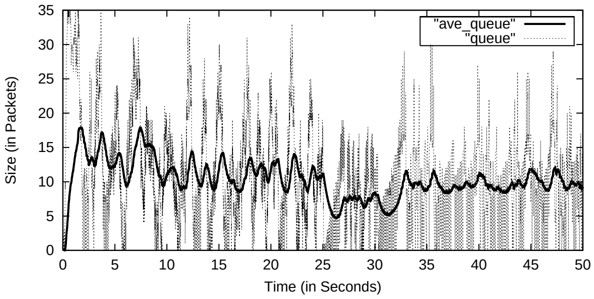

Figure 9: Adaptive RED with a Decrease in Congestion.

Every $interval$ seconds:
    if ($avg > target$ and $max_p \leq 0.5$)
        increase max_p:
        $max_p \leftarrow max_p + \alpha$;
    elseif ($avg < target$ and $max_p \geq 0.01$)
        decrease max_p:
        $max_p \leftarrow max_p * \beta$;

Variables:
avg: average queue size

Fixed parameters:
$interval$: time; 0.5 seconds
$target$: target for avg;
    $[min_{th} + 0.4 * (max_{th} - min_{th})$,
        $min_{th} + 0.6 * (max_{th} - min_{th})]$.
α: increment; min(0.01, max_p/4)
β: decrease factor; 0.9

Figure 10: The Adaptive RED algorithm.

queue size—to dominate on smaller time scales. The adaptation of max_p is invoked only as needed over longer time scales.

The robustness of Adaptive RED comes from its slow and infrequent adjustments of max_p. The price of this slow modification is that after a sharp change in the level of congestion, as in Figures 7 and 9, it could take some time, possibly ten or twenty seconds, before max_p adapts to its new value. To ensure that the performance of Adaptive RED will not be unduly degraded during this transition period, our third guideline restricts max_p to stay within the range [0.01, 0.5]. This ensures that during the transition period the overall performance of RED should still be acceptable, even though the average queue size might not be in its target range, and the average delay or throughput might suffer slightly.

We do not claim that our algorithm for Adaptive RED is optimal, or even close to optimal, but it seems to work well in a wide range of scenarios, and we believe that it could safely be deployed now in RED implementations in the Internet. As a result of the slow adaptation of max_p, the design of Adaptive RED gives robust performance in a wide range of environments. As stated above, the cost of this slow adaptation is that of a transient period, after a sharp change in the level of congestion, when the average queue size is not within the target zone. Adaptive RED is thus consciously positioned in the conservative, robust end of the spectrum of AQM mechanisms, with the aim of avoiding the more finely-tuned but also more fragile dynamics at the more aggressive end of the spectrum.

Adaptive RED's algorithm in Figure 10 uses AIMD to adapt max_p. While we experimented with other linear controls such as MIMD (Multiplicative Increase Multiplicative Decrease) as well as proportional error controls, as might be suggested by some control-theoretic analyses, our experiences have been that the AIMD approach is more robust.

This completes the general description of the Adaptive RED algorithm. Embedded in this algorithm are detailed choices for various parameters. We now briefly justify these choices.

4.1 The range for max_p

The upper bound of 0.5 on max_p can be justified on two grounds. First, we are not trying to optimize RED for packet drop rates greater than 50%. In addition, because we use RED in gentle mode, this means that the packet drop rate varies from 1 to max_p as the average queue size varies from min_{th} to max_{th}, and the packet drop rate varies from max_p to 1 as the average queue size varies from max_{th} to twice max_{th}. Thus, with max_p set to 0.5, the packet drop rate varies from 0 to 1 as the average queue size varies from min_{th} to twice max_{th}. This should give somewhat robust performance even with packet drop rates greater than 50%. The upper bound of 0.5 on max_p means that when the packet drop rate exceeds 25%, the average queue size could exceed the target range by up to a factor of four.[4]

The lower bound of 0.01 on max_p is motivated by a desire to limit the range of max_p. We believe that for scenarios with very small packet drop rates, RED will perform fairly robustly with max_p set to 0.01, and no one is likely to object to an average queue size less than the target range.

4.2 The parameters α and β

We note that it takes at least $0.49/\alpha$ intervals for max_p to increase from 0.01 to 0.50; this is 24.5 seconds for our default parameters for α and $interval$ (see Figure 10). Similarly, it takes at least $\log 0.02/\log\beta$ intervals for max_p to decrease from 0.50 to 0.01; with our default parameters, this is 20.1 seconds. Given a sharp change from one level of congestion to another, 25 seconds is therefore an upper bound on the interval during which the average queue size could be outside its target range, and the performance of the AQM might be somewhat degraded.

In recommending values for α and β, we want to ensure that under normal conditions a single modification of max_p does not result in the average queue size moving from above the target range to below it, or vice versa. Let's assume for simplicity that when max_p is adapted the steady-state packet dropping probability p remains the same, and the average queue size avg simply shifts to match the new value of max_p. Thus, assuming $p < max_p$, when max_p increases by α, avg can be expected to decrease from $min_{th} + \frac{p}{max_p}(max_{th} - min_{th})$ to $min_{th} + \frac{p}{max_p+\alpha}(max_{th} - min_{th})$. This is a decrease of

$$\frac{\alpha}{(max_p+\alpha)}\frac{p}{max_p}(max_{th} - min_{th}).$$

As long as this is less than $0.2(max_{th} - min_{th})$, the average queue size should not change from above the target range to below the target range in a single interval. This suggests choosing $\frac{\alpha}{(max_p+\alpha)} < 0.2$, or equivalently, $\alpha < 0.25\ max_p$. Our default setting of α (shown in Figure 10) obeys this constraint.

Similarly, we have to check that the mutliplicative decrease of max_p does not cause the average queue size to go from below to above the target range after a single adjustment of max_p. A similar analysis to that for α shows that as long as

$$\frac{p(1-\beta)}{max_p\beta}(max_{th} - min_{th}) < 0.2(max_{th} - min_{th}),$$

the average queue size should not change from below the target range to above the target range in a single interval. This suggests choosing $\frac{1-\beta}{\beta} < 0.2$, or equivalently, $\beta > 0.83$. This constraint is satisfied by our default value of 0.9 for β (see Figure 10).

4.3 Setting RED parameters max_{th} and w_q

As described above, Adaptive RED removes RED's dependence on the parameter max_p. To reduce the need for other parameter-tuning for RED, we also specify procedures for automatically setting the RED parameters max_{th} and w_q.

In automatic mode max_{th} is set to three times min_{th}. This follows the recommendations in [9].[5] In this case the target average queue size is centered around $2 \times min_{th}$, and is therefore determined only by the RED parameter min_{th}. Considerations in specifying the target average queue size are discussed in Section 6.

The guidelines for setting w_q given in the original RED paper [12] are in terms of the transient queue size accommodated by RED, and the time required by the estimator to respond to a step change in the actual queue size. From [12], if the queue size changes from one value to another, it takes $-1/\ln(1-w_q)$ packet arrivals for the average queue to reach 63% of the way to the new value. Thus, we refer to $-1/\ln(1-w_q)$ as the "time constant" of the estimator for the average queue size, even though this "time constant" is specified in packet arrivals and not in time itself.

The default in the NS simulator is for w_q to be set to 0.002; this corresponds to a time constant of 500 packet arrivals. However, for a 1 Gbps link with 500-byte packets, 500 packet arrivals corresponds to a small fraction of a

[4] For $max_{th} = k\ min_{th}$, the target queue size is $\frac{k+1}{2}min_{th}$, and with packet drop rates approaching 100% and max_p set to 50%, the average queue size approaches $2max_{th} = 2k\ min_{th}$.

[5] By chance, we haven't followed this recommendation in all of the simulations in this paper, but following the recommendation does not change our results.

round-trip time (1/50-th of an assumed round-trip time of 100 ms). Clearly higher-speed links require smaller values for w_q, so that the time constant remains on the order of round-trip times, rather than fractions of round-trip times. Following the approachs in [15, 21], in automatic mode we set w_q as a function of the link bandwidth.

For RED in automatic mode, we set w_q to give a time constant for the average queue size estimator of one second; this is equivalent to ten round-trip times, assuming a default round-trip time of 100 ms. Thus, we set

$$w_q = 1 - exp(-1/C) \tag{1}$$

where C is the link capacity in packets/second, computed for packets of the specified default size.

5 Simulations

The simulations in Section 3 suggest that Adaptive RED, by automatically setting w_q and continually adapting max_p, achieves the goals of high throughput and low average queueing delays across a wide variety of conditions. In this section we more closely examine three aspects Adaptive RED's behavior: oscillations, effects of w_q, and response to routing dynamics.

5.1 Exploring Oscillations

Because of the feedback nature of TCP's congestion control, oscillations in the queue length are very common. Some oscillations are "malignant", in that they degrade overall throughput and increase variance in queuing delay; other oscillations are "benign" oscillations and do not significantly effect either throughput or delay. Figures 11 through 14 each show the average queue size for a simulation with 100 long-lived flows, each with a round-trip time of 250ms, and with a congested link of 15Mbps. All of the flows use ECN and 1000-byte data packets. The RED queue management has $min_{th} = 20$ and $max_{th} = 80$.

The simulation in Figure 11, which uses RED, has three factors that each encourage oscillations in the queue size: (1) a fixed (and overly small) value for max_p; (2) a high value for w_q; and (3) a simple traffic mix of one-way traffic of long-lived flows. Figure 11 shows dramatic oscillations in the average queue size, with the average queue size going below min_{th} and above max_{th} in each oscillation. This leads to oscillations between periods of high packet drop rates and periods of no packet drops, and results in degraded throughput and high variance in queueing delay. Exceeding max_{th} incurs a non-linearity in the form of a large packet drop, with a corresponding decrease in utilization, and forces the average queue size to decrease sharply, in this case, below min_{th}. But when the average queue size falls below min_{th}, the average packet drop probability becomes zero, and the flows once again ramp up their congestion windows over the next few RTTs, thereby sustaining the oscillations. In this case, RED achieves a link utilization of 90%, and a high variance in queuing delay. The packet loss rate is about 3.5%, even with the use of ECN.

Figure 12 shows that such malignant oscillations are significantly dampened with a more realistic traffic mix including reverse path traffic and web traffic; that is, even with a badly tuned and non-adaptive RED, many of the worst effects of the oscillations are decreased when a slightly more realistic traffic load is used.

We now consider how Adaptive RED, with its lower value for w_q and its automatically adapting max_p, fares in these two traffic scenarios. Figure 13 shows that even with the simple traffic mix of one-way, long-lived traffic, Adaptive RED is able to eliminate the malignant oscillations, and turn them into benign oscillations. Thus, in spite of the high, fixed RTT of 250ms and with neither reverse traffic nor web traffic, Adaptive RED achieves a utilization of 96.8%, an average queuing queue size oscillating within the target range of [44, 56] packets, and a negligable loss rate. (Recall that the traffic is using ECN.) We note that malignant oscillations persist with Adaptive RED if the larger value of 0.002 is used for w_q; that is, adapting max_p and choosing a good value for w_q are both required for eliminating the malignant oscillations for this scenario.

Figure 14 shows that with a slightly more realistic traffic mix, with web traffic and reverse traffic, the benign oscillations of Figure 13 have been replaced by more irregular variations of the average queue size, generally within the target range of [44, 56] packets. The utilization in this case is also slightly higher than that in Figure 13.

5.2 The Effects of Queue Weight

While Figure 1 showed the performance costs, in terms of decreased throughput, of too large a value for w_q, this section illustrates the costs, in terms of increased queuing delay, of too small a value for w_q.

Figures 15 through 17 show the results of a simple simulation with two long-lived TCP flows, each with a round-trip around 45ms, competing over a 15Mbps link. The second TCP flow starts a time 2.5 in a 10-second simulation. With two TCP flows, the average congestion window should be around 85 packets. All three simulations use non-adaptive RED, and differ only in w_q. These simulations also illustrate one of the costs of an overly-small value of w_q, of being slow to respond to a large, sustained increase in the

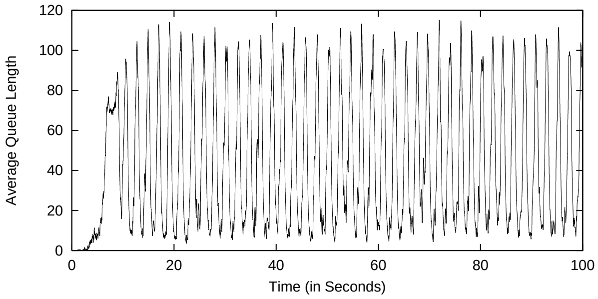

Figure 11: RED, one-way long-lived traffic, w_q=0.002.

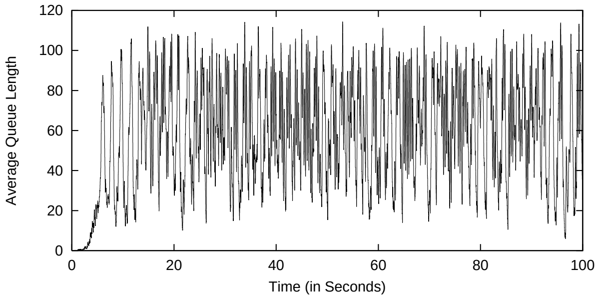

Figure 12: RED, richer traffic mix, w_q=0.002.

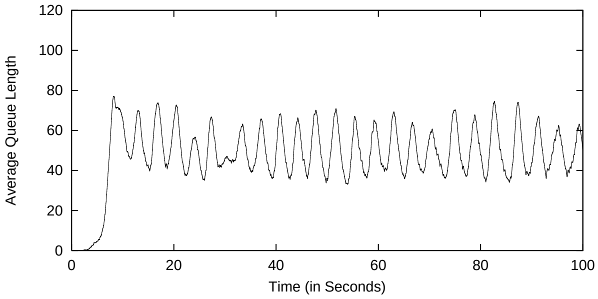

Figure 13: Adaptive RED, one-way long-lived traffic, w_q=0.00027.

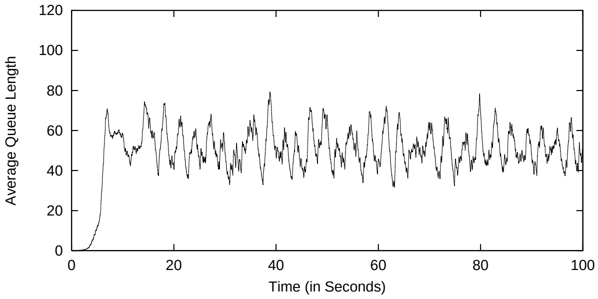

Figure 14: Adaptive RED, richer traffic mix, w_q=0.00027.

instantaneous queue.

The simulation in Figure 15 uses RED with a large value for w_q of 0.002. All of the simulations use the automatic settings for min_{th} and max_{th}, resulting in min_{th} set to 19 packets and max_{th} set to $3min_{th}$. Figure 15 shows the instantaneous queue size, as well as the average queue size estimated by RED. Although the second TCP is cut off in its slow-start slightly before it reaches its desired congestion window in Figure 15, Figures 15 and 16 both show reasonable good performance for this scenario.

In contrast, Figure 17 shows one of the costs of having w_q set too small. In this simulation $w_q = 0.0001$, with the result that RED is slow to detect a sudden increase in congestion, and does not detect the congestion building up at time 2.5 until a queue of 350 packets has built up. In this simulation the sharp increase in the queue is due to the slow-start of a single high-bandwidth TCP flow, but the increase could also have been due to a flash crowd, a routing failure, or a denial-of-service attack. This simulation is run with a large buffer size, allowing the spike in the queue to reach 350 packets. If the buffer size had been smaller, then the simulation would simply have reverted to the typical behavior of Drop-Tail queue management, of multiple packets dropped from a window of data. We explored a range of scenarios, and in almost all cases, even in steady-state scenarios, the link utilization suffered when we used a smaller value of w_q than suggested by Equation 1.

5.3 Simulations of Routing Changes

This section explores briefly the transient behavior of Adaptive RED in environments with sharp changes in the load due to routing changes. Figure 18 illustrates the average queue size as a function of time in a simulation where the output link becomes unavailable from time 50 to time 60 (in seconds). The simulation topology includes an alternate path with a lower precedence but only half the link capacity, so the TCP connections continue to send packets during the link outage. When the link comes back up, the entire load is shifted back to the original link. The link utilization reaches 88.3% over the 10-second period immediately following the repair, and 96.1% for the following 10-second period. Thus, this scenario demonstrates the good dynamic behavior of Adaptive RED. More extensive results are presented in [10].

6 Tradeoffs between Throughput and Delay

Given the Adaptive RED algorithm and the automatic setting of maxthresh and w_q described earlier in this paper, the only critical parameter left to specify for RED is the target average queue size. Adaptive RED maintains an average queue size of twice minthresh; therefore, given a target for the average queue size, setting minthresh is straightforward. The hard part is determining the desired average queue size.

The "optimal" average queue size for a router is a function of the relative tradeoff between throughput and delay,

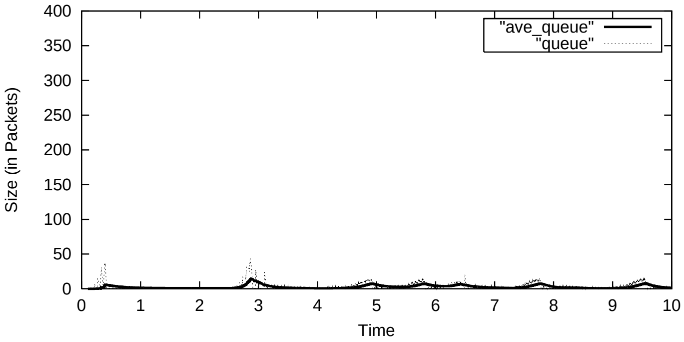

Figure 15: RED, two flows, w_q too large, at 0.002.

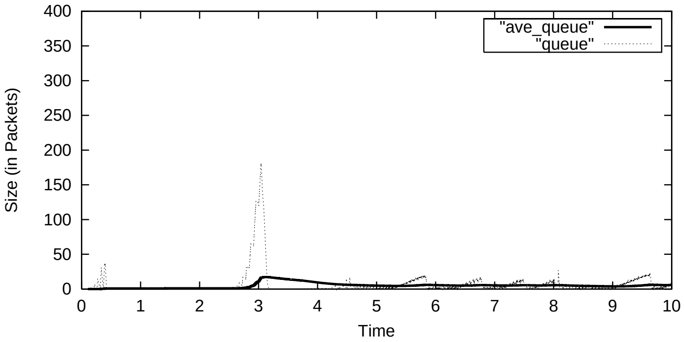

Figure 16: RED, automatic setting for w_q, 0.00027.

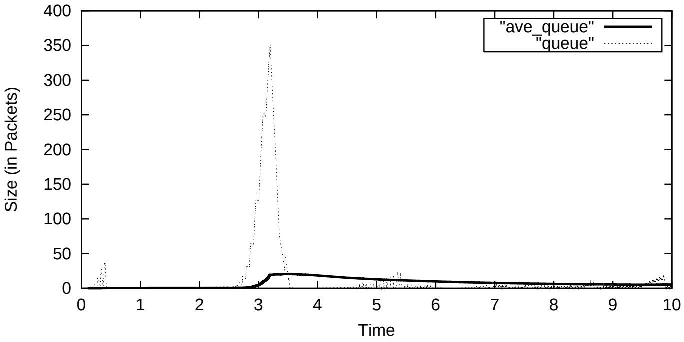

Figure 17: RED, w_q too small, at 0.0001.

and this tradeoff is necessarily a question of policy. In addition, these tradeoffs between throughput and delay are a function of characteristics of the aggregate traffic, in particular of the burstiness of the aggregate. Thus, scenarios with one-way traffic, long-lived flows, short RTTs, and a high level of statistical multiplexing allow both very high throughput and very low delay, while scenarios with higher burstiness that results from two-way traffic and web mice or scenarios with low levels of statistical multiplexing require some harder tradeoffs between throughput and delay.

Leaving behind the issue of optimality, and following Jacobson et al. in [15] and the simulation scripts in [11], in automatic mode we set min_{th} as a function of the link bandwidth. For slow and moderate speed links, we have found that setting min_{th} to five packets works well, so we continue to use this as a lower bound for min_{th} in automatic mode. For a high-speed link, however, an average queue size of ten packets is very small relative to the delay-bandwidth product, and results in a severe loss of throughput.

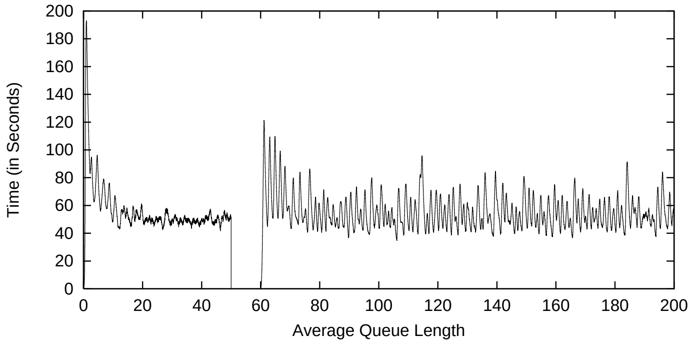

Figure 18: Average queue size variation with routing change.

Our rule of thumb for a plausible tradeoff between throughput and delay is to require that the average queuing delay at a router be only a fraction of the end-to-end round-trip time, using a default value of 100ms. Setting $min_{th} = \frac{delay_{target}\ C}{2}$ gives a target average queuing delay of $delay_{target}$ seconds, for C the link capacity in pkts/sec. We use a $delay_{target}$ of 5ms, and in automatic mode we set min_{th} to $Max\left[5, \frac{delay_{target}\ C}{2}\right]$ packets. This translates to setting minthresh to 12.5 packets for a 10Mbps link, and to 125 packets for a 100Mbps link. [10] reports on extensive simulations exploring the tradeoffs between throughput and delay in a range of settings.

7 Related Work

The parameter sensitivity of RED has been discussed in a number of papers, and we briefly discuss some of this related work in this section. There is a growing body of research on AQM, and our paper builds upon observations from a range of this earlier work. We do not attempt to evaluate each of these proposals here, but simply note that we don't believe that any of these proposals has yet presented the full answer to the parameter sensitivity of RED. In particular, we believe that none of these proposals have presented a deployable mechanism for adapting the RED parameter max_p. Similarly, for the proposals not based upon RED, we do not believe that any of these has yet provided a robust, deployable proposal for AQM for realistic scenarios with bursty two-way traffic and a range of packet sizes. Some of these proposals will be evaluated in a separate work.

The Adaptive RED proposal in this paper is based on the original Adaptive RED proposed by Feng *et al.*, in [6, 7], of adapting max_p as a function of the average queue size. This original Adaptive RED adjusts the packet dropping probability, max_p, in RED to keep the average queue size greater than minthresh and less than maxthresh. In particular, the original version of Adaptive RED increased max_p multiplicatively when the average queue size went below minthresh, and decreased max_p multiplicatively when the average queue size went above maxthresh.

Jacobson *et al.* in [15], a early draft of an in-progress paper, suggest a self-tuning RED with the RED parameters determined by the bandwidth of the output link. Other proposals in [15] include setting the average queue size to the instantaneous queue size whenever the instantaneous size is less than the average, and setting minthresh to $0.3P$.

Ziegler *et al.*[21, 22, 23] explore the stability of RED, and recommend settings for max_p so that the average queue size converges to $\frac{min_{th}+max_{th}}{2}$. In this paper, we follow their goal of loosely converging to a certain queue size: "we define convergence very loosely as achieving a state of bounded oscillation of the queue-size around a value between min_{th} and max_{th} so that the amplitude of the oscillation of the average queue size is significantly smaller than $max_{th} - min_{th}$ and the instantaneous queue-size remains greater than zero and smaller than the total buffersize" [21]. [21] recommends settings of min_{th}, max_{th}, w_q, and max_p to achieve these goals. Ziegler *et al.* set w_q as a function of the link bandwidth to give a fixed time constant in of one second for the estimator. Ziegler *et al.* also show that the original Adaptive RED from [6, 7] does not always give good performance.

May *et al.*'s critical evaluation of RED in [18] summarizes as follows: "RED with small buffers does not improve significantly the performance of the network", and "parameter tuning in RED remains an inexact science, but has no big impact on end-to-end performance".

Christiansen *et al.*[5] evaluated RED experimentally in a laboratory scenario with web traffic with congestion only in the forward path, and concluded that RED offers no clear advantage over tail-drop FIFO in terms of per-connection response times for web users. The paper also reports that performance is quite sensitive to the setting of RED parameters.

Misra et at. in [19] also discuss the difficulties in tuning RED parameters. They illustrate the benign oscillations in the instantaneous queue size, and say that they are currently investigating tuning RED parameters. Hollot et al. in [13] also focus on oscillations in the queue size, and use this starting point to recommend values for RED parameters.

Firoiu et al. in [8] also considered problems with RED such as oscillations in the queue size, and made recommendations for configuring RED parameters. [8] recommended that the ideal rate for sampling the average queue size is once per round-trip time.

A number of papers have proposed alternate mechanisms for active queue management. These include Ott et al.'s Stabilized RED (SRED) [20], Lapsley et al.'s Random Early Marking (REM) [17, 4], Hollot et al.'s Proportional-Integral (PI) controller [14], and Kunniyur et al.'s AVG [16].Several of these proposals share Adaptive RED's goal of keeping a stable average queue size with changing levels of congestion. AVG tries to keep the average queue size small even during high congestion, and uses a token bucket with a fill rate less than the link capacity. The PI controller is primarily designed to avoid oscillations in the queue size. SRED tackles the goal of stabilizing the buffer occupancy by estimating the number of active flows. Aweya et al.'s Dynamic-RED (DRED) [2, 3] also has the goal of maintaining the queue size close to a threshold value, and uses a controller that adapts the packet-dropping probability as a function of the average distance of the queue from the threshold value.

8 Conclusions

In this paper we have reported on Adaptive RED, which, by adapting the RED parameter max_p and automatically setting the RED parameters w_q and max_{th}, maintains a predictable average queue size and reduces RED's parameter sensitivity. Adaptive RED, however, leaves the choice of the target queue size to network operators who must make a policy tradeoff between utilization and delay. In future work, we plan to explore the use of Adaptive RED in virtual queues with the goal of providing very small average queueing delays. In this case, the virtual queue would be configured with a throughput slightly lower than the actual throughput of the link so that the queuing delay would be determined solely by the traffic burstiness.

References

[1] J. Aweya, M. Ouellette, D. Y. Montuno and A. Chapman. Enhancing TCP Performance with a Load-adaptive RED Mechanism. *International Journal of Network Management*, V. 11, N. 1, 2001

[2] J. Aweya, M. Ouellette, D. Y. Montuno and A. Chapman. A Control Theoretic Approach to Active Queue Management. *Computer Networks 36*, 2001.

[3] J. Aweya, M. Ouellette, D. Y. Montuno and A. Chapman. An Optimization-oriented View of Random

Early Detection. *Computer Communications*, 2001, to appear.

[4] S. Athuraliya, D. Lapsley, and S. Low. An Enhanced Random Early Marking Algorithm for Internet Flow Control. *Infocom* 2000.

[5] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. Tuning RED for Web Traffic. *SIGCOMM*, pages 139–150, Sep. 2000.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin. Techniques for Eliminating Packet Loss in Congested TCP/IP Network. U. Michigan CSE-TR-349-97, November 1997.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. *Infocom*, Mar 1999.

[8] Victor Firoiu and Marty Borden. A Study of Active Queue Management for Congestion Control. *Infocom*, pages 1435–1444, 2000.

[9] S. Floyd. RED: Discussions of Setting Parameters, November 1997. http://www.aciri.org/floyd/REDparameters.txt.

[10] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED. Technical Report, to appear, 2001.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TFRC Web Page, 2000. http://www.aciri.org/tfrc/.

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

[13] C. Hollot, V. Misra, D. Towsley, and W. Gong. A Control Theoretic Analysis of RED. *Infocom*, 2001.

[14] C. Hollot, V. Misra, D. Towsley, and W. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. *Infocom*, 2001.

[15] V. Jacobson, K. Nichols, and K. Poduri, RED in a Different Light, September 1999. draft, www.cnaf.infn.it/˜ferrari/papers/ispn/red_light_9_30.pdf.

[16] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. CSL Technical Report, University of Illinois, January 2001.

[17] D. Lapsley and S. Low. Random Early Marking for Internet Congestion Control. *Proceedings of Globecom '99*, pages 1747–1752, December 1999.

[18] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons Not to Deploy RED. *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99)*, pages 260–262, June 1999.

[19] Vishal Misra, Wei-Bo Gong, and Donald F. Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *SIGCOMM*, pages 151–160, 2000.

[20] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. *Infocom*, 1999.

[21] T. Ziegler, S. Fdida, and C. Brandauer. Stability Criteria for RED with Bulk-data TCP Traffic, 2001. Technical Report, August 1999. http://www-rp.lip6.fr/publications/production.html.

[22] T. Ziegler, S. Fdida, and C. Brandauer. Stability Criteria for RED with TCP Traffic, May 2000. Technical Report, http://www.newmedia.at/ tziegler/papers.html.

[23] T. Ziegler, S. Fdida, C. Brandauer, and B. Hechenleitner. Stability of RED with Two-way TCP Traffic, October 2000. *IEEE ICCCN*, http://www.newmedia.at/ tziegler/papers.html.