Random Early Detection Gateways for Congestion Avoidance

Sally Floyd and Van Jacobson*

Lawrence Berkeley Laboratory
University of California
floyd@ee.lbl.gov
van@ee.lbl.gov

To appear in the August 1993 IEEE/ACM Transactions on Networking

Abstract

This paper presents Random Early Detection (RED) gateways for congestion avoidance in packet-switched networks. The gateway detects incipient congestion by computing the average queue size. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or by setting a bit in packet headers. When the average queue size exceeds a preset threshold, the gateway drops or *marks* each arriving packet with a certain probability, where the exact probability is a function of the average queue size.

RED gateways keep the average queue size low while allowing occasional bursts of packets in the queue. During congestion, the probability that the gateway notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the gateway. RED gateways are designed to accompany a transport-layer congestion control protocol such as TCP. The RED gateway has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their window at the same time. Simulations of a TCP/IP network are used to illustrate the performance of RED gateways.

1 Introduction

In high-speed networks with connections with large delay-bandwidth products, gateways are likely to be designed with correspondingly large maximum queues to accommodate transient congestion. In the current Internet, the TCP transport protocol detects congestion only after a packet has been dropped at the gateway. However, it would clearly be undesirable to have large queues (possibly on the order of a delay-bandwidth product) that were full much of the time; this would significantly increase the average delay in the network. Therefore, with increasingly high-speed networks, it is increasingly important to have mechanisms that keep throughput high but average queue sizes low.

In the absence of explicit feedback from the gateway, there are a number of mechanisms that have been proposed for transport-layer protocols to maintain high throughput and low delay in the network. Some of these proposed mechanisms are designed to work with current gateways [15, 23, 31, 33, 34], while other mechanisms are coupled with gateway scheduling algorithms that require per-connection state in the gateway [20, 22]. In the absence of explicit feedback from the gateway, transport-layer protocols could infer congestion from the estimated bottleneck service time, from changes in throughput, from changes in

---

*This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

end-to-end delay, as well as from packet drops or other methods. Nevertheless, the view of an individual connection is limited by the timescales of the connection, the traffic pattern of the connection, the lack of knowledge of the number of congested gateways, the possibilities of routing changes, as well as by other difficulties in distinguishing propagation delay from persistent queueing delay.

The most effective detection of congestion can occur in the gateway itself. The gateway can reliably distinguish between propagation delay and persistent queueing delay. Only the gateway has a unified view of the queueing behavior over time; the perspective of individual connections is limited by the packet arrival patterns for those connections. In addition, a gateway is shared by many active connections with a wide range of roundtrip times, tolerances of delay, throughput requirements, etc.; decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself.

The method of monitoring the average queue size at the gateway, and of notifying connections of incipient congestion, is based of the assumption that it will continue to be useful to have queues at the gateway where traffic from a number of connections is multiplexed together, with FIFO scheduling. Not only is FIFO scheduling useful for sharing delay among connections, reducing delay for a particular connection during its periods of burstiness [4], but it scales well and is easy to implement efficiently. In an alternate approach, some congestion control mechanisms that use variants of Fair Queueing [20] or hop-by-hop flow control schemes [22] propose that the gateway scheduling algorithm make use of per-connection state for every active connection. We would suggest instead that per-connection gateway mechanisms should be used *only* in those circumstances where gateway scheduling mechanisms without per-connection mechanisms are clearly inadequate.

The DECbit congestion avoidance scheme [18], described later in this paper, is an early example of congestion detection at the gateway; DECbit gateways give explicit feedback when the average queue size exceeds a certain threshold. This paper proposes a different congestion avoidance mechanism at the gateway, RED (Random Early Detection) gateways, with somewhat different methods for detecting congestion and for choosing which connections to notify of this congestion.

While the principles behind RED gateways are fairly general, and RED gateways can be useful in controlling the average queue size even in a network where the transport protocol can not be trusted to be cooperative, RED gateways are intended for a network where the transport protocol responds to congestion indications from the network. The gateway congestion control mechanism in RED gateways simplifies the congestion control job required of the transport protocol, and should be applicable to transport-layer congestion control mechanisms other than the current version of TCP, including protocols with rate-based rather than window-based flow control.

However, some aspects of RED gateways are specifically targeted to TCP/IP networks. The RED gateway is designed for a network where a single marked or dropped packet is sufficient to signal the presence of congestion to the transport-layer protocol. This is different from the DECbit congestion control scheme, where the transport-layer protocol computes the *fraction* of arriving packets that have the congestion indication bit set.

In addition, the emphasis on avoiding the global synchronization that results from many connections reducing their windows at the same time is particularly relevant in a network with 4.3-Tahoe BSD TCP [14], where each connection goes through Slow-Start, reducing the window to one, in response to a dropped packet. In the DECbit congestion control scheme, for example, where each connection's response to congestion is less severe, it is also less critical to avoid this global synchronization.

RED gateways can be useful in gateways with a range of packet-scheduling and packet-dropping algorithms. For example, RED congestion control mechanisms could be implemented in gateways with drop preference, where packets are marked as either "essential" or "optional", and "optional" packets are dropped first when the queue exceeds a certain size. Similarly, for a gateway with separate queues for realtime and non-realtime traffic, for example, RED congestion control mechanisms could be applied to the queue for one of these traffic classes.

The RED congestion control mechanisms monitor the average queue size for each output queue, and, using randomization, choose connections to notify of that congestion. Transient congestion is accommodated by a temporary increase in the queue. Longer-lived congestion is reflected by an increase in the computed average queue size, and results in randomized feedback to some of the connections to decrease their windows. The probability that a connection is notified of congestion is proportional to that connection's share of the throughput through the gateway.

Gateways that detect congestion before the queue overflows are not limited to packet drops as the method for notifying connections of congestion. RED gateways can *mark* a packet by dropping it at the gateway or by setting a bit in the packet header, depending on the transport protocol. When the average queue size exceeds a maximum threshold, the RED gateway marks every packet that arrives at the gateway. If RED gateways mark packets by *dropping* them, rather than by setting a bit in the packet header, when the average queue size exceeds the maximum threshold, then the RED gateway controls the average queue size even in the absence of a cooperating transport protocol.

One advantage of a gateway congestion control mechanism that works with current transport protocols, and that does not require that all gateways in the internet use the same gateway congestion control mechanism, is that it could be deployed gradually in the current Internet. RED gateways are a simple mechanism for congestion avoidance that could be implemented gradually in current TCP/IP networks with no changes to transport protocols.

Section 2 discusses previous research on Early Random Drop gateways and other congestion avoidance gateways. Section 3 outlines design guidelines for RED gateways. Section 4 presents the RED gateway algorithm, and Section 5 describes simple simulations. Section 6 discusses in detail the parameters used in calculating the average queue size, and Section 7 discusses the algorithm used in calculating the packet-marking probability.

Section 8 examines the performance of RED gateways, including the robustness of RED gateways for a range of traffic and for a range of parameter values. Simulations in Section 9 demonstrate, among other things, the RED gateway's lack of bias against bursty traffic. Section 10 describes how RED gateways can be used to identify those users that are using a large fraction of the bandwidth through a congested gateway. Section 11 discusses methods for efficiently implementing RED gateways. Section 12 gives conclusions and describes areas for future work.

2 Previous work on congestion avoidance gateways

2.1 Early Random Drop gateways

Several researchers have studied Early Random Drop gateways as a method for providing congestion avoidance at the gateway.[1]

Hashem [11] discusses some of the shortcomings of Random Drop[2] and Drop Tail gateways, and briefly investigates Early Random Drop gateways. In the implementation of Early Random Drop gateways in [11], if the queue length exceeds a certain *drop level*, then the gateway drops each packet arriving at the gateway with a fixed *drop probability*. This is discussed as a rough initial implementation. Hashem [11] stresses that

---

[1] Jacobson [14] proposed gateways to monitor the average queue size to detect incipient congestion, and to randomly drop packets when congestion is detected. These proposed gateways are a precursor to the Early Random Drop gateways that have been studied by several authors [11] [36]. We refer to the gateways in this paper as Random Early Detection or RED gateways. RED gateways differ from the earlier Early Random Drop gateways in several respects: the *average* queue size is measured; the gateway is not limited to *dropping* packets; and the packet-marking probability is a function of the average queue size.

[2] With Random Drop gateways, when a packet arrives at the gateway and the queue is full, the gateway randomly chooses a packet from the gateway queue to drop.

in future implementations the drop level and the drop probability should be adjusted dynamically, depending on network traffic.

Hashem [11] points out that with Drop Tail gateways each congestion period introduces global synchronization in the network. When the queue overflows, packets are often dropped from several connections, and these connections decrease their windows at the same time. This results in a loss of throughput at the gateway. The paper shows that Early Random Drop gateways have a broader view of traffic distribution than do Drop Tail or Random Drop gateways and reduce global synchronization. The paper suggests that because of this broader view of traffic distribution, Early Random Drop gateways have a better chance than Drop Tail gateways of targeting aggressive users. The conclusions in [11] are that Early Random Drop gateways deserve further investigation.

For the version of Early Random Drop gateways used in the simulations in [36], if the queue is more than half full then the gateway drops each arriving packet with probability 0.02. Zhang [36] shows that this version of Early Random Drop gateways was not successful in controlling misbehaving users. In these simulations, with both Random Drop and Early Random Drop gateways, the misbehaving users received roughly 75% higher throughput than the users implementing standard 4.3 BSD TCP.

The Gateway Congestion Control Survey [21] considers the versions of Early Random Drop described above. The survey cites the results in which the Early Random Drop gateway is unsuccessful in controlling misbehaving users [36]. As mentioned in [32], Early Random Drop gateways are not expected to solve all of the problems of unequal throughput given connections with different roundtrip times and multiple congested gateways. In [21], the goals of Early Random Drop gateways for congestion avoidance are described as "uniform, dynamic treatment of users (streams/flows), of low overhead, and of good scaling characteristics in large and loaded networks". It is left as an open question whether or not these goals can be achieved.

2.2 Other approaches to gateway mechanisms for congestion avoidance

Early descriptions of IP Source Quench messages suggest that gateways could send Source Quench messages to source hosts before the buffer space at the gateway reaches capacity [26], and before packets have to be dropped at the gateway. One proposal [27] suggests that the gateway send Source Quench messages when the queue size exceeds a certain threshold, and outlines a possible method for flow control at the source hosts in response to these messages. The proposal also suggests that when the gateway queue size approaches the maximum level the gateway could discard arriving packets other than ICMP packets.

The DECbit congestion avoidance scheme, a binary feedback scheme for congestion avoidance, is described in [29]. In the DECbit scheme the gateway uses a *congestion-indication* bit in packet headers to provide feedback about congestion in the network. When a packet arrives at the gateway, the gateway calculates the average queue length for the last (busy + idle) period plus the current busy period. (The gateway is *busy* when it is transmitting packets, and *idle* otherwise.) When the average queue length exceeds one, then the gateway sets the congestion-indication bit in the packet header of arriving packets.

The source uses window flow control, and updates its window once every two roundtrip times. If at least half of the packets in the last window had the congestion indication bit set, then the window is decreased exponentially. Otherwise, the window is increased linearly.

There are several significant differences between DECbit gateways and the RED gateways described in this paper. The first difference concerns the method of computing the average queue size. Because the DECbit scheme chooses the last (busy + idle) cycle plus the current busy period for averaging the queue size, the queue size can sometimes be averaged over a fairly short period of time. In high-speed networks with large buffers at the gateway, it would be desirable to explicitly control the time constant for the computed average queue size; this is done in RED gateways using time-based exponential decay. In [29] the authors report that they rejected the idea of a weighted exponential running average of the queue length because when the time interval was far from the roundtrip time, there was bias in the network. This problem of bias does not arise with RED gateways because RED gateways use a randomized algorithm for marking packets, and assume that the sources use a different algorithm for responding to marked packets. In a DECbit network, the source looks at the fraction of packets that have been marked in the last roundtrip time. For a network with RED gateways, the source should reduce its window even if there is only one marked packet.

A second difference between DECbit gateways and RED gateways concerns the method for choosing connections to notify of congestion. In the DECbit scheme there is no conceptual separation between the algorithm to detect congestion and the algorithm to set the congestion indication bit. When a packet arrives at the gateway and the computed average queue size is too high, the congestion indication bit is set in the header of that packet. Because of this method for marking packets, DECbit networks can exhibit a bias against bursty traffic [see Section 9]; this is avoided in RED gateways by using randomization in the method for marking packets. For congestion avoidance gateways designed to work with TCP, an additional motivation for using randomization in the method for marking packets is to avoid the global synchronization that results from many TCP connections reducing their window at the same time. This is less of a concern in networks with the DECbit congestion avoidance scheme, where each source decreases its window fairly moderately in response to congestion.

Another proposal for adaptive window schemes where the source nodes increase or decrease their windows according to feedback concerning the queue lengths at the gateways is presented in [25]. Each gateway has an upper threshold UT indicating congestion, and a lower threshold LT indicating light load conditions. Information about the queue sizes at the gateways is added to each packet. A source node increases its window only if all the gateway queue lengths in the path are below the lower thresholds. If the queue length is above the upper threshold for any queue along the path, then the source node decreases its window. One disadvantage of this proposal is that the network responds to the instantaneous queue lengths, not to the average queue lengths. We believe that this scheme would be vulnerable to traffic phase effects and to biases against bursty traffic, and would not accommodate transient increases in the queue size.

3 Design guidelines

This section summarizes some of the design goals and guidelines for RED gateways. The main goal is to provide congestion avoidance by controlling the average queue size. Additional goals include the avoidance of global synchronization and of a bias against bursty traffic and the ability to maintain an upper bound on the average queue size even in the absence of cooperation from transport-layer protocols.

The first job of a congestion avoidance mechanism at the gateway is to detect incipient congestion. As defined in [18], a *congestion avoidance* scheme maintains the network in a region of low delay and high throughput. The average queue size should be kept low, while fluctuations in the actual queue size should be allowed to accommodate bursty traffic and transient congestion. Because the gateway can monitor the size of the queue over time, the gateway is the appropriate agent to detect incipient congestion. Because the gateway has a unified view of the various sources contributing to this congestion, the gateway is also the appropriate agent to decide which sources to notify of this congestion.

In a network with connections with a range of roundtrip times, throughput requirements, and delay sensitivities, the gateway is the most appropriate agent to determine the size and duration of short-lived bursts in queue size to be accommodated by the gateway. The gateway can do this by controlling the time constants used by the low-pass filter for computing the average queue size. The goal of the gateway is to detect incipient congestion that has persisted for a “long time” (several roundtrip times).

The second job of a congestion avoidance gateway is to decide which connections to notify of congestion at the gateway. If congestion is detected before the gateway buffer is full, it is not necessary for the gateway to drop packets to notify sources of congestion. In this paper, we say that the gateway *marks* a packet, and *notifies* the source to reduce the window for that connection. This marking and notification can consist of dropping a packet, setting a bit in a packet header, or some other method understood by the transport protocol. The current feedback mechanism in TCP/IP networks is for the gateway to drop packets, and the simulations of RED gateways in this paper use this approach.

One goal is to avoid a bias against bursty traffic. Networks contain connections with a range of burstiness, and gateways such as Drop Tail and Random Drop gateways have a bias against bursty traffic. With Drop Tail gateways, the more bursty the traffic from a particular connection, the more likely it is that the gateway queue will overflow when packets from that connection arrive at the gateway [7].

Another goal in deciding which connections to notify of congestion is to avoid the global synchronization that results from notifying all connections to reduce their windows at the same time. Global synchronization has been studied in networks with Drop Tail gateways [37], and results in loss of throughput in the network. Synchronization as a general network phenomena has been explored in [8].

In order to avoid problems such as biases against bursty traffic and global synchronization, congestion avoidance gateways can use distinct algorithms for congestion detection and for deciding which connections to notify of this congestion. The RED gateway uses randomization in choosing which arriving packets to mark; with this method, the probability of marking a packet from a particular connection is roughly proportional to that connection's share of the bandwidth through the gateway. This method can be efficiently implemented without maintaining per-connection state at the gateway.

One goal for a congestion avoidance gateway is the ability to control the average queue size even in the absence of cooperating sources. This can be done if the gateway *drops* arriving packets when the average queue size exceeds some maximum threshold (rather than setting a bit in the packet header). This method could be used to control the average queue size even if most connections last less than a roundtrip time (as could occur with modified transport protocols in increasingly high-speed networks), and even if connections fail to reduce their throughput in response to marked or dropped packets.

4 The RED algorithm

This section describes the algorithm for RED gateways. The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a *minimum* threshold and a *maximum* threshold. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold.

When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability p_a, where p_a is a function of the average queue size avg. Each time that a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the gateway. The general RED gateway algorithm is given in Figure 1.

Thus the RED gateway has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion. The goal is for the gateway to mark packets at fairly evenly-spaced intervals, in order to avoid biases and to avoid global synchronization, and to mark packets sufficiently frequently to control the average queue size.

The detailed algorithm for the RED gateway is given in Figure 2. Section 11 discusses efficient implementations of these algorithms.

The gateway's calculations of the average queue size take into account the period when the queue is

```
for each packet arrival
    calculate the average queue size $avg$
    if $min_{th} \leq avg < max_{th}$
        calculate probability $p_a$
        with probability $p_a$:
            mark the arriving packet
    else if $max_{th} \leq avg$
        mark the arriving packet
```

Figure 1: General algorithm for RED gateways.

empty (the idle period) by estimating the number m of small packets that *could* have been transmitted by the gateway during the idle period. After the idle period the gateway computes the average queue size as if m packets had arrived to an empty queue during that period.

As avg varies from min_{th} to max_{th}, the packet-marking probability p_b varies linearly from 0 to max_p:

$$p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th}).$$

The final packet-marking probability p_a increases slowly as the count increases since the last marked packet:

$$p_a \leftarrow p_b/(1 - count \cdot p_b)$$

As discussed in Section 7, this ensures that the gateway does not wait too long before marking a packet.

The gateway marks each packet that arrives at the gateway when the average queue size avg exceeds max_{th}.

One option for the RED gateway is to measure the queue in bytes rather than in packets. With this option, the average queue size accurately reflects the average delay at the gateway. When this option is used, the algorithm would be modified to ensure that the probability that a packet is marked is proportional to the packet size in bytes:

$$\begin{aligned} p_b &\leftarrow max_p(avg - min_{th})/(max_{th} - min_{th}) \\ p_b &\leftarrow p_b \, \text{PacketSize}/\text{MaximumPacketSize} \\ p_a &\leftarrow p_b/(1 - count \cdot p_b) \end{aligned}$$

In this case a large FTP packet is more likely to be marked than is a small TELNET packet.

Sections 6 and 7 discuss in detail the setting of the various parameters for RED gateways. Section 6 discusses the calculation of the average queue size. The queue weight w_q is determined by the size and duration of bursts in queue size that are allowed at the gateway. The minimum and maximum thresholds min_{th} and max_{th} are determined by the desired average queue size. The average queue size which makes the desired tradeoffs (such as the tradeoff between maximizing throughput and minimizing delay) depends on network characteristics, and is left as a question for further research. Section 7 discusses the calculation of the packet-marking probability.

In this paper our primary interest is in the functional operation of the RED gateways. Specific questions about the most efficient implementation of the RED algorithm are discussed in Section 11.

5 A simple simulation

This section describes our simulator and presents a simple simulation with RED gateways. Our simulator is a version of the REAL simulator [19] built on Columbia's Nest simulation package [1], with extensive

```
Initialization:
    $avg \leftarrow 0$
    $count \leftarrow -1$
for each packet arrival
    calculate the new average queue size $avg$:
        if the queue is nonempty
            $avg \leftarrow (1-w_q)avg + w_q\,q$
        else
            $m \leftarrow f(time - q\_time)$
            $avg \leftarrow (1-w_q)^m avg$
    if $min_{th} \leq avg < max_{th}$
        increment $count$
        calculate probability $p_a$:
            $p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th})$
            $p_a \leftarrow p_b/(1 - count \cdot p_b)$
        with probability $p_a$:
            mark the arriving packet
            $count \leftarrow 0$
    else if $max_{th} \leq avg$
        mark the arriving packet
        $count \leftarrow 0$
    else $count \leftarrow -1$
when queue becomes empty
    $q\_time \leftarrow time$
```

Saved Variables:

avg: average queue size
q_time: start of the queue idle time
$count$: packets since last marked packet

Fixed parameters:

w_q: queue weight
min_{th}: minimum threshold for queue
max_{th}: maximum threshold for queue
max_p: maximum value for p_b

Other:

p_a: current packet-marking probability
q: current queue size
$time$: current time
$f(t)$: a linear function of the time t

Figure 2: Detailed algorithm for RED gateways.

modifications and bug fixes made by Steven McCanne at LBL. In the simulator, FTP sources always have a packet to send and always send a maximal-sized (1000-byte) packet as soon as the congestion control window allows them to do so. A sink immediately sends an ACK packet when it receives a data packet. The gateways use FIFO queueing.

Source and sink nodes implement a congestion control algorithm equivalent to that in 4.3-Tahoe BSD TCP.[3] Briefly, there are two phases to the window-adjustment algorithm. A threshold is set initially to half the receiver's advertised window. In the slow-start phase, the current window is doubled each roundtrip time until the window reaches the threshold. Then the congestion-avoidance phase is entered, and the current window is increased by roughly one packet each roundtrip time. The window is never allowed to increase to more than the receiver's advertised window, which this paper refers to as the "maximum window size". In 4.3-Tahoe BSD TCP, packet loss (a dropped packet) is treated as a "congestion experienced" signal. The source reacts to a packet loss by setting the threshold to half the current window, decreasing the current window to one packet, and entering the slow-start phase.

Figure 3 shows a simple simulation with RED gateways. The network is shown in Figure 4. The simulation contains four FTP connections, each with a maximum window roughly equal to the delay-bandwidth product, which ranges from 33 to 112 packets. The RED gateway parameters are set as follows: $w_q = 0.002$, $min_{th} = 5$ packets, $max_{th} = 15$ packets, and $max_p = 1/50$. The buffer size is sufficiently large that packets are never dropped at the gateway due to buffer overflow; in this simulation the RED gateway controls the average queue size, and the actual queue size never exceeds forty packets.

For the charts in Figure 3, the x-axis shows the time in seconds. The bottom chart shows the packets from nodes 1-4. Each of the four main rows shows the packets from one of the four connections; the bottom row shows node 1 packets, and the top row shows node 4 packets. There is a mark for each data packet as it arrives at the gateway and as it departs from the gateway; at this time scale, the two marks are often indistinguishable. The y-axis is a function of the packet sequence number; for packet number n from node i, the y-axis shows $n \bmod 90 + (i-1)100$. Thus, each vertical 'line' represents 90 consecutively-numbered packets from one connection arriving at the gateway. Each 'X' shows a packet dropped by the gateway, and each 'X' is followed by a mark showing the retransmitted packet. Node 1 starts sending packets at time 0, node 2 starts after 0.2 seconds, node 3 starts after 0.4 seconds, and node 4 starts after 0.6 seconds.

The top chart of Figure 3 shows the instantaneous queue size q and the calculated average queue size avg. The dotted lines show min_{th} and max_{th}, the minimum and maximum thresholds for the average queue size. Note that the calculated average queue size avg changes fairly slowly compared to q. The bottom row of X's on the bottom chart shows again the time of each dropped packet.

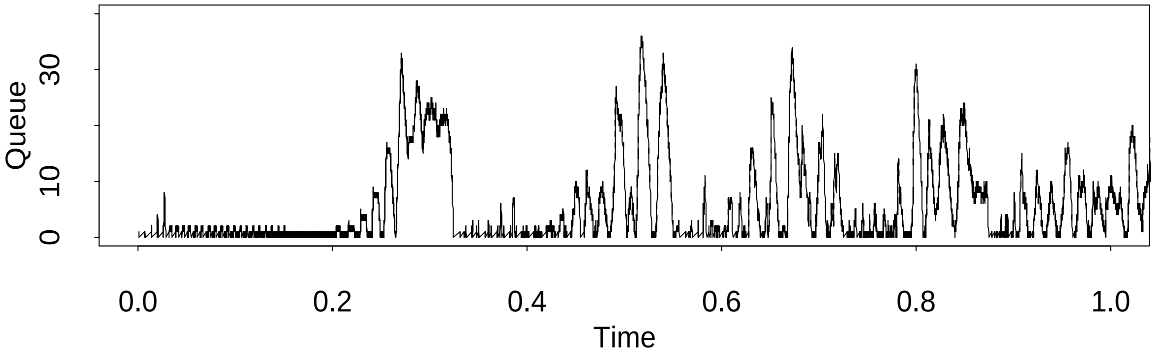
This simulation shows the success of the RED gateway in controlling the average queue size at the gateway in response to a dynamically changing load. As the number of connections increases, the frequency with which the gateway drops packets also increases. There is no global synchronization. The higher throughput for the connections with shorter roundtrip times is due to the bias of TCP's window increase algorithm in favor of connections with shorter roundtrip times (as discussed in [6, 7]). For the simulation in Figure 3 the average link utilization is 76%. For the following second of the simulation, when all four sources are active, the average link utilization is 82%. (This is not shown in Figure 3.)

Because RED gateways can control the average queue size while accommodating transient congestion, RED gateways are well-suited to provide high throughput and low *average* delay in high-speed networks with TCP connections that have large windows. The RED gateway can accommodate the short burst in the queue required by TCP's slow-start phase; thus RED gateways control the *average* queue size while still allowing TCP connections to smoothly open their windows. Figure 5 shows the results of simulations of the network in Figure 6 with two TCP connections, each with a maximum window of 240 packets, roughly equal to the delay-bandwidth product. The two connections are started at slightly different times. The simulations compare the performance of Drop Tail and of RED gateways.

In Figure 5 the x-axis shows the total throughput as a fraction of the maximum possible throughput on the congested link. The y-axis shows the average queue size in packets (as seen by arriving packets).

---

[3] Our simulator does not use the 4.3-Tahoe TCP code directly but we believe it is functionally identical.



Queue size (solid line) and average queue size (dashed line).