2RegionRED: a Congestion Control Mechanism for the High Speed Internet

Karen Wang

December 19, 2001

This research was supported by ARPA Agreement J958100
under Contract No. F30602-00-20553.

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

2RegionRED: a Congestion Control Mechanism for the High Speed Internet

by

Karen Wang

Abstract

This thesis proposes a new Active Queue Management (AQM) scheme called 2RegionRED. It is superior to the classic Random Early Detection (RED) algorithm in that there is an intuitive way to set its parameters and it is self-tuning. Its design is motivated by an original principle to sustain the smallest queue possible while still allowing for maximum link utilization. 2RegionRED uses the number of competing TCPs as its measure of load. However it does not keep an explicit count. The result is a novel algorithm that adjusts the drop rate according to two regions of operation: that requiring less than and greater than one drop per round-trip time (RTT). This thesis also analyzes methods for measuring the persistent queue and proposes the ABSMIN method. Simulations of 2RegionRED using ABSMIN reveal some difficulties and insights. Basic comparisons to the Adaptive RED and Flow Proportional Queuing (FPQ) adaptive algorithms are also demonstrated through simulation.

Keywords: 2RegionRED, adaptive, congestion control, algorithms, Internet, RED, Adaptive RED, FPQ, queue estimation, EWMA, and ABSMIN.

This report is a minor revision of the Master Thesis of the same title submitted to the Department of Electrical Engineering and Computer Science on December 14, 2001, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering. The thesis was supervised by Dr. David D. Clark, Senior Research Scientist at the Laboratory for Computer Science.

Acknowledgments

My thesis advisor David D. Clark for his mentorship and example in how to do research and how to think on my own, yet inspiring me all the time with his own ideas and suggestions; for his intuitions that I have come to respect and for always startling me by his sharp ability to point out the subtle yet revealing things in my plots that escape my notice; for his patience, and for making me realize how fun and fulfilling research can be; finally, for inspiring me by his heterogeneous engineering qualities– his involvement in the policy side of things as well because, in his words, "I care about the Internet".

My officemate Xiaowei Yang for her encouragement, for all the fun conversations networking related and otherwise, and the use of her debugged version of NS TCP Full code for simulating little background Web mice.

Dorothy Curtis for her kind generosity in always helping me get set me up on fast machines or finding the memory to run my simulations.

Timothy Shepard for demonstrating lots of nifty features in his xplot plotting tool which helped me to verify that various transport protocols were operating according to their specs.

Dina Katabi for sharing her insight and expertise on Internet congestion control.

Wenjia Fang for answering many questions about RIO (RED with In and Out bit) and for her original NS RIO code.

The Advanced Network Architecture (ANA) group here at the Laboratory for Computer Science (LCS) which has got me thinking about the big picture of things.

My sister Amy and friend Michelle Crotwell for their prayers and for feeding me, and Michelle for housing me in the last months.

My fiance Jen-Guan Li for being a constant source of encouragement and joy during these past two and a half years at MIT, for making many visits to Cambridge, Mass., and for often staying up late just to work beside me, distract me, and keep me company via the Internet videophone.

My parents for their love and support, for working so incredibly hard so that Amy and I could have an easier life and all the opportunities we might wish for, and for wishing only that we be happy and healthy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Need for Active Queue Management

In the 1980's, which marked the early days of the Internet, a primitive version of the Transmission Control Protocol (TCP) was used to provide connections with reliable in-order delivery of packets. Sometime thereafter the Internet began to exhibit a form of degenerate behavior termed *congestion collapse*, or Internet meltdown, in which the network becomes very busy but little useful work is done. As a consequence *congestion avoidance* mechanisms were added to TCP, as a means of controlling congestion from the edges of the network [19]. These mechanisms, described in more detail in Section 2.2, have been largely responsible for preventing recurrences of Internet meltdown.

As the Internet has developed and grown, however, it has become apparent that the degree to which the edges of the network can control congestion is limited [4]. The popular solution is to complement the end-to-end mechanisms by additional mechanisms embedded in routers at the core of the network.

A router is equipped with a *queue*, or *buffer*, that can hold some maximum number of packets. There is a class of router congestion control algorithms called *Queue Management* algorithms (in contrast to scheduling algorithms) [4] whose goal is to control the length of this queue by determining the appropriate rate at which packets should be dropped. As discussed in the next section, dropping packets is effective in controlling queue size because dropped packets are inferred by their TCP senders as signs of congestion in the network. If these TCP senders are well-behaved, they will respond by cutting down their sending rates to help alleviate the situation.

The traditional router queue management technique is called *tail drop*. Tail drop controls congestion simply by dropping all newly arriving packets to the queue when the buffer overflows. The idea, however, behind so-called *Active Queue Management* (AQM) schemes, is to drop packets *before* the buffer overflows. That is, such schemes aim to detect the onset of congestion and remedy the situation early without sacrificing network performance. The AQM scheme that has received much attention in recent years is the Random Early Detection (RED) algorithm [16]. RED operates by using the average queue size as a congestion indicator, with a higher average queue indicating more severe congestion and calling for more aggressive dropping.

1.2 AQM and TCP: Two Control Loops

We saw above how the actions taken by TCP and the AQM scheme at the router are not independent. If we consider RED as the AQM scheme, we find that the interaction between TCP and RED is better understood when RED is viewed as a closed control loop [31, 20]. Furthermore, the RED control loop within the router may be embedded in a larger end-to-end loop, depending on the traffic type. (See Figure 1-1.) In the case of TCP traffic, actions taken by the inner RED control loop may be greatly magnified by the outer loop.

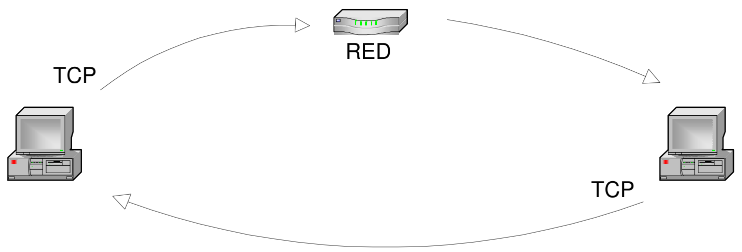


Figure 1-1: Two Control loops: End to end (TCP), and router's AQM (RED).

Suppose an intermediate RED router along the path responds to a growing queue by dropping a packet. The outer loop, characterized by TCP's end-to-end congestion control mechanisms, reacts to the drop by slowing down the source (halving its window), an action which also effectively reduces the queue size at the congested router. Thus, the outer loop works together with the inner RED loop.

In the following chapters we will see how many AQM schemes, including the one we propose, are designed with end-to-end TCP dynamics in mind. In this report we choose to consider TCP traffic alone because it is the most prevalent kind of traffic on the Internet today.

1.3 The Problem with RED

The original RED AQM scheme, because of its popularity, has also come under much scrutiny. One of its shortcomings is attributed to the static setting of its parameters. A set of RED parameters which works well for one particular traffic load or traffic mix may lead to less than satisfactory performance for another. In other words, static parameters limit the range of scenarios which RED can handle. As a result, some circumstances require parameters to be periodically and manually reset. In order to avoid this extremely unappealing idea, RED needs to adapt.

Another drawback of RED is that the tuning of its parameters remains an imprecise science. There are some general guidelines on what has worked well, but not a very concrete intuition of how parameters should be set and why some parameter settings "work" and why others do not given a particular load or traffic mix. This is an important question because the sensitivity of certain RED parameters require the user to have a good understanding of the algorithm. Parameter sensitivity increases the likelihood that poorly set parameters will lead to an unacceptable degradation in the algorithm's performance. For these reasons, RED either needs to offer some clearer rationale on how to set its parameters, or it must be made more robust.

1.4 Contribution of this Thesis

In this thesis, we analyze some principles which can guide the design of a workable RED algorithm. The fruit of this effort is embodied in an algorithm called 2RegionRED.

2RegionRED is so named because it distinguishes two regions of operation. The two regions of operation refer to regions in which the drop rate necessary to "control" the system is less than or greater than one drop per round-trip time (RTT). The appropriate region of operation is dependent on the load, which 2RegionRED associates with the number of competing flows (N) through a bottleneck. When there are few flows, a small drop rate is sufficient, and the algorithm is said to be operating in the "Low-N Region". More aggressive dropping is required in the "High-N Region" where the number of competing flows is large. This scheme differs from RED, which uses queue length as an indication of load.

Though the idea of correlating load with the number of flows is not new, the mechanism that 2RegionRED uses to discern the proper region of operation is unique. In 2RegionRED, a design choice was made which separates the Low-N and High-N regions into distinct physical portions of the buffer. Thus, the size of the queue at any moment determines the region of operation. Note that the algorithm does *not* explicitly count the number of flows passing through the router. Rather, N is implicitly derived.

Requiring less than or greater than one drop per RTT affects the mechanism that 2RegionRED chooses to enforce the drop rate. The Low-N Region is driven by a single deterministic drop each time the queue size reaches a predetermined level. This level is chosen in a simple manner according to a delay-utilization tradeoff. In the High-N Region, 2RegionRED homes in on an appropriate drop rate by looking at correlations between a current drop rate and the subsequent *change* in the queue size. From these correlations N can be implicitly derived and used to adjust the drop rate. We believe this approach has not been used elsewhere.

2RegionRED proves to be adaptive to changing traffic loads. There is a logical way to set the algorithm's parameters, which eliminates guesswork, and requires very little manual tuning. By associating load with the number of flows rather than the queue length, there is a very clear rationale behind what the proper discard rate should be, as well as clear rationale behind the resulting design of 2RegionRED.

In addition to 2RegionRED, this thesis also suggests and analyzes a mechanism called ABSMIN for measuring the persistent queue. How the queue is estimated plays a critical role in determining the size the AQM scheme perceives the effective queue to be, which in turn determines how the AQM scheme reacts. We find that especially when considering the burstiness of bi-directional traffic, ABSMIN is superior to both the EWMA (Exponentially Weighted Moving Average) scheme used in RED and a variant we call EWMA' in tracking the persistent queue.

1.5 Organization of this Thesis

Chapter 2 covers background on TCP. Chapter 3 covers background and related work on RED. Chapter 4 discusses the newly proposed algorithm, 2RegionRED, and the principles that guide its design. Chapter 5 studies the ABSMIN method for measuring the persistent queue and compares its performance to EWMA and EWMA'. Chapter 6 looks at the performance of 2RegionRED (using ABSMIN queue estimation) in simulation with comparison to some other proposed adaptive algorithms. Finally, Chapter 7 concludes with mention of interesting future work.

Chapter 2

TCP Dynamics: Points of Interest

It is important to understand the dynamics of TCP. Its behavior impacts the design of various AQM schemes including 2RegionRED, the algorithm proposed in Chapter 4. Understanding TCP behavior is also critical to understanding and assessing the performance and limitations of 2RegionRED. Note that the Reno version of TCP is used throughout this thesis and in the simulations. [1]

2.1 TCP Intro

As of 2001, the dominant transport protocol in the Internet is currently the Transmission Control Protocol (TCP). Today, Internet applications that desire reliable data delivery use TCP. TCP guarantees an in-order reliable bytestream. TCP segments are assigned sequence numbers. [2]

Reliability is achieved through the use of acknowledgments, or ACKs. When a TCP receiver receives a datagram, an ACK packet is generated and sent back to the source. This ACK identifies the sequence number of the next packet it is expecting. Note that ACKs are cumulative, which means that if packets 1 through 10 arrive back-to-back at a receiver,

[1] There are many different versions of TCP: Tahoe, Reno, NewReno, SACK, Vegas. Modern day implementations are leaning towards NewReno and SACK. The use of Reno, which is fundamentally the same as NewReno, is sufficient for our purposes in deriving the appropriate behavior of 2RegionRED in later chapters.

[2] For clarity, this thesis refers to "packets" instead of segments as the unit of data transfer.

generating a single ACK with sequence number 11 effectively acknowledges them all.

TCP is a window based protocol and operates via sliding window. `Cwnd`, the congestion window size of each TCP, limits the number of unacknowledged bytes or packets that the TCP sender can have in transit. As packets are acknowledged, the window "slides" across the sequence number space, allowing more packets to be transmitted. This is what is referred to by *ACK-clocking*.

Fast Retransmit If a packet is lost before reaching the TCP receiver, the TCP sender is able to detect the loss via duplicate ACKs. That is, as packets following the lost packet arrive at the destination, ACKs are generated, each containing the same sequence number – that of the lost packet. When TCP receives three duplicate ACKs, it takes this as a sign that there was congestion somewhere along its path, and retransmits the packet. This is called *Fast Retransmit*. Note that standard TCP assumes that links are not lossy, such that in most cases a lost packet is an indication of congestion, rather than packet corruption which is not as uncommon on a wireless link.

Costly Timeouts, Fast Recovery Note that when a packet is lost, a TCP sender's window becomes "locked" in place, with the left edge of the window held at the sequence number of the earliest lost unacknowledged packet. Once the TCP has sent all the packets that its window allows, it blocks waiting for the ACK for the lost packet. In some cases, there is a point where there are not enough packets left in the pipe to generate the three duplicate ACKs needed to trigger a fast retransmit. This often occurs when a TCP connection experiences multiple packet losses (depending upon which version of TCP is used), or if its window is too small. In this case, TCP relies on a retransmit timer. After a certain length of time without seeing an ACK, usually on the order of a couple hundred milliseconds, the timer will go off and the earliest unacknowledged packet will be retransmitted. When this occurs, we say that a *timeout* has occurred. Timeouts are undesirable as the long idle period before the timer goes off can be extremely costly to connection throughput. The *Fast Recovery* mechanism that was added to the TCP protocol temporarily inflates a TCP's window on the receipt of many duplicate ACKs. This has the effect of preventing the ACK stream from running dry for a little while longer in effort to avoid some unnecessary timeouts.

2.2 TCP Congestion Control Mechanisms

In addition to Fast Retransmit and Fast Recovery, two other congestion control mechanisms were added to TCP that have been largely responsible for the stability of the Internet for over the past decade. These are *Slow Start* and *Congestion Avoidance* [1].

2.2.1 Slow Start

TCP's congestion control mechanisms involve two main variables: `cwnd` and `ssthresh`.

`Cwnd`, as mentioned earlier, is the congestion window size of each TCP sender. It limits the number of unacknowledged bytes or packets that the TCP sender can have in transit at any time. Without this limitation, the TCP sender could send a large burst of data (limited by the receiver's advertised window) all at once. Large bursts are undesirable as they cause limited network buffers to overflow, resulting in dropped packets usually belonging to various TCP connections. This could lead to synchronization and in the worst case congestion collapse.

When a TCP connection starts, it is in *slow start phase*. It has no knowledge of the state of the network, and of what rate it can send packets into the network. Rather than sending a random burst of packets into the network, the TCP sender probes the network for available bandwidth. It does this by opening its congestion window in the following manner. Upon startup, a TCP sender will begin with a small initial value for the congestion window `cwnd`. Every round-trip time, it will increase its sending rate by at most a factor of two until congestion is detected, either from a packet loss, or an ECN congestion notification. Thus, in slow start phase, `cwnd` grows exponentially. When congestion is detected, the probing terminates and `ssthresh` is set to the highest value of `cwnd` before the congestion was detected.

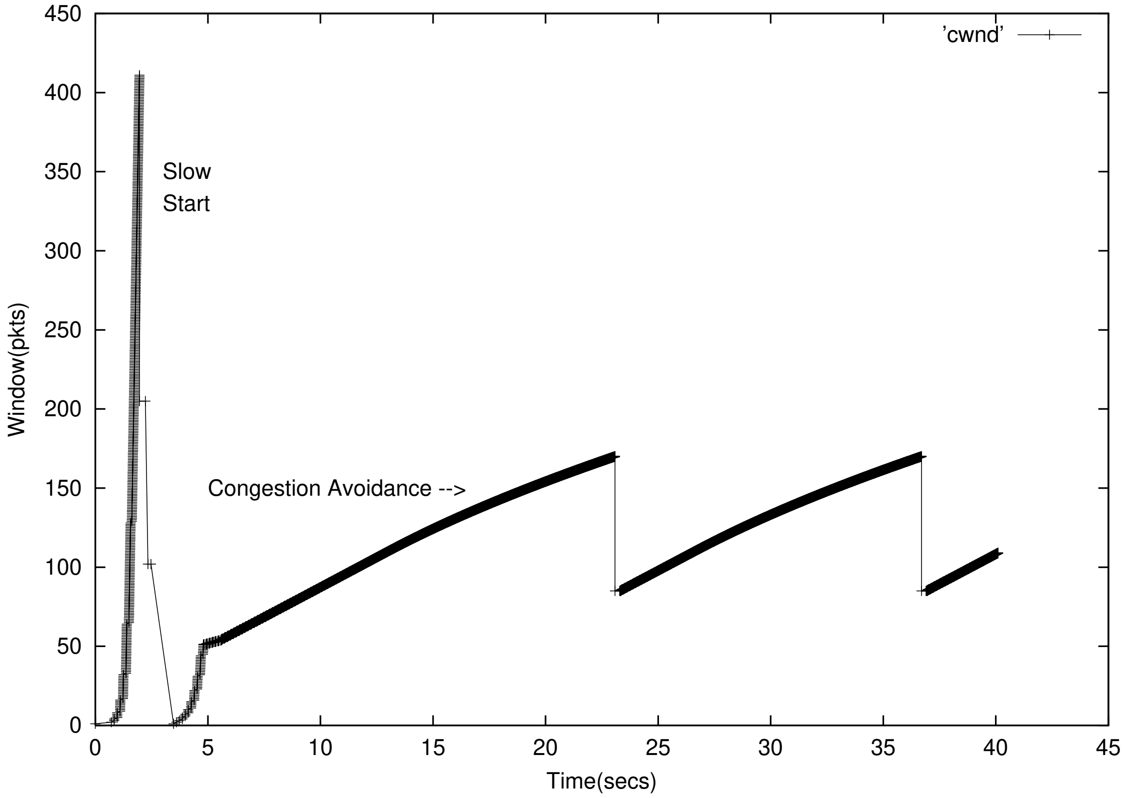


Figure 2-1: Slow Start and Congestion Avoidance.

2.2.2 Congestion Avoidance

After slow start completes, a TCP enters into the *congestion avoidance* phase, where it hunts for a reasonable window size. Above `ssthresh`, `cwnd` is incremented by at most one packet every round-trip time, so that its window increases linearly (as opposed to exponentially in the slow-start phase). [3]

In response to a congestion indication, both `cwnd` and `ssthresh` are halved. The idea behind this is to increase the window slowly, in case more network bandwidth has been made available, and to decrease its share of network resources if bandwidth appears to be tight. `Cwnd` then resumes its increment-by-one behavior per round-trip time. Thus, this period of congestion avoidance is exemplified by a "sawtooth" pattern, and is also described as having Additive Increase, Multiplicative Decrease (AIMD) behavior.

[3] Actually, in practice, `cwnd` is maintained in bytes. Rather than wait for an entire windows' worth of packets before incrementing `cwnd`, TCP increments `cwnd` by a fraction of a packet upon the arrival of each ACK [29, pages 407-408].

2.3 The consequences of ACK-Compression

ACK-compression is an unusual phenomena. It was originally discovered in simulations [32] and its existence has been identified in real networks. Its most obvious trademark is a rapidly fluctuating queue length, which will be of concern when we assess the performance of the 2RegionRED algorithm described in Chapter 4. It is important to understand ACK-Compression, as it affects the behavior and performance of a network, as later simulations will reveal.

ACK-compression results from an interaction between data and ACK packets when there is two-way traffic and appears to require only two assumptions: "(1) ACK packets are significantly smaller than data packets (2) the packets from different connections are clustered" [32, page 8].

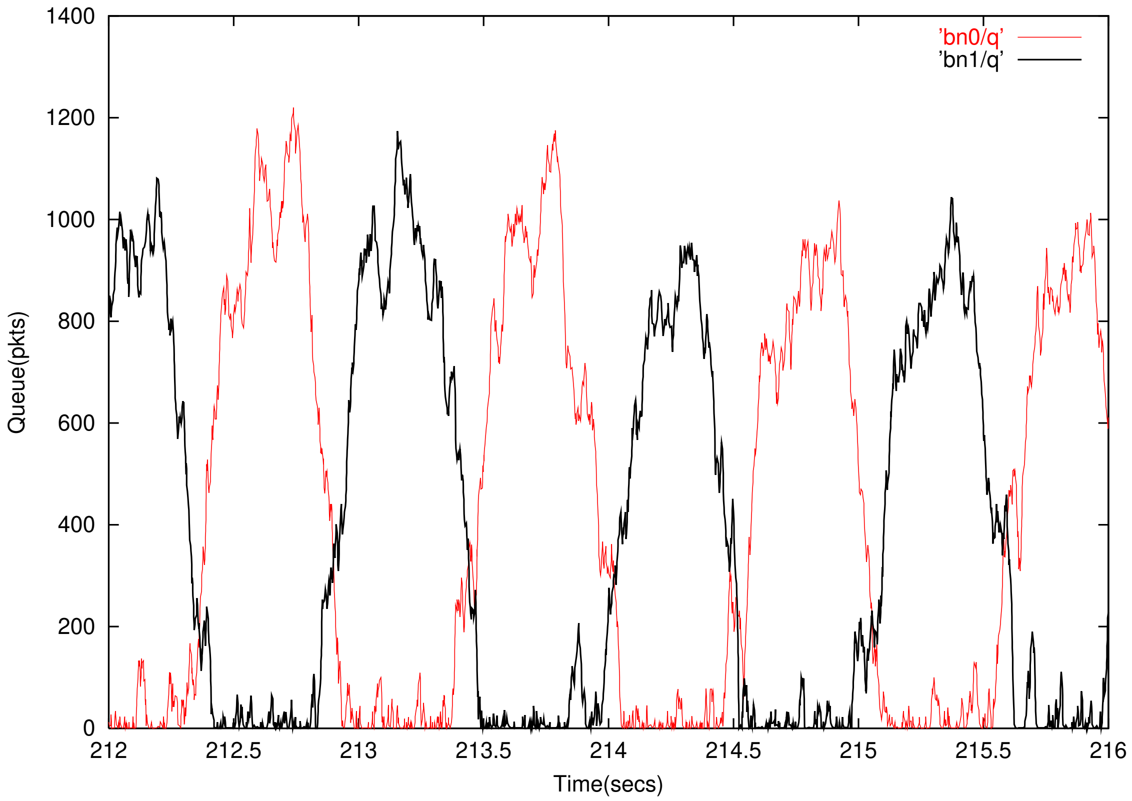


Figure 2-2: Example of ACK-Compression

TCP is window-based and operates via an ACK-clocking mechanism. Because in normal circumstances, another packet is sent immediately upon the receipt of an ACK, the packets in an entire window become more or less clustered or clumped together. Thus, TCP exhibits the clustering in assumption (2). That ACK packets are much smaller than data packets, as assumed in (1), is generally true.

At the source, data packets are spaced by the transmission time of the data packet. At the receiver, an ACK is generated immediately upon the receipt of a data packet, so the spacing of packets is preserved and ACKs are separated according to the transmission time of a data packet. When ACKs encounter a congested router, however, they are queued. This causes them to lose their spacing (hence the term "compressed"). Upon leaving the queue, ACKs are separated by the transmission time of an ACK packet, which is smaller according to assumption (1). This disrupts the ACK-clocking, and is the key observation that triggers the ACK-compression phenomena.

One significant consequence of ACK-compression is that "... the number of packets that can be in flight at any one time depends on how many compressed ACK's are in the pipe. Thus, there is no longer a well-defined capacity C that can reliably predict the occurrence of the congestion epochs." [32, page 17] The utilization suffers in both directions. A classic example of ACK-compression is depicted in Figure 2-2, which displays the queue size in both directions on a link.

2.4 Delay-bandwidth Product

The *delay-bandwidth product* or *pipesize* (P) of a link is a useful metric. Bandwidth (B) is the capacity, in packets per second, on a particular link interface. Delay (D) is the average round-trip time, in seconds, of TCP connections traversing the link. The product (D*B) indicates the total number of packets that can be in flight in both directions of the link at any instant.

In this thesis, the delay-bandwidth product is also used to set the buffer limit. It is a general rule of thumb to allocate enough buffer space to hold a pipesize worth of packets.

Chapter 3

AQM Background and Related Work

Congestion control in the Internet is a very broad topic. There are many flavors of different congestion control mechanisms. This chapter covers background on the Random Early Detection (RED) [16] algorithm and related congestion control algorithms.

One taxonomy classifies different congestion control mechanisms for best-effort service according to four criteria: packet scheduling, buffer management, feedback, and end adjustments, all of which it deems necessary and sufficient [21]. According to this taxonomy, RED is classified as both an indirect method of scheduling control and a queue management mechanism, with implicit feedback and a slow start end adjustment algorithm.

There is another taxonomy for congestion control algorithms in packet switching networks that is similar but based on control theory [31]. Under this taxonomy, RED is classified as having closed loop control, (global) implicit feedback, and a slow start end adjustment algorithm. A number of recent papers take this control theoretic approach to analyzing RED [3, 20, 18].

The studies in this thesis focus upon this class of algorithms [1] , and RED in particular.

[1] To be precise, however, these classifications are really descriptions of the RED queue management strategy as coupled with the TCP end adjustment algorithm. It is interesting to observe how RED is designed with end system dynamics in mind. This was elaborated in Section 1.2.

RED parameter	function
Bmin (min_{th})	minimum buffer threshold
Bmax (max_{th})	maximum buffer threshold
Pmax (max_p)	dropping probability at Bmax
buffer size	hard queue limit
w_q	gain for computing the EWMA of the queue length

Table 3.1: RED parameters

3.1 RED Algorithm

The Random Early Detection (RED) algorithm [16] has been the focus of much attention recently. This algorithm may be referred to as *traditional* or *classic RED* throughout this paper. RED emphasizes that it is able to avoid global synchronization and alleviate biases of bursty sources.

RED involves the setting of several parameters, which are preset in the router. Some of the parameter names used in this paper differ slightly from the original terminology, and is used for clarity. These are displayed in Table 3.1. The original terminology is shown in parentheses, if different.

A RED gateway detects incipient congestion by monitoring the average queue size. This average queue size is calculated according to the following equation, an exponentially-weighted moving average (EWMA) of the instantaneous queue length:

$$aveq \longleftarrow (1 - w_q) * aveq + w_q * q \tag{3.1}$$

$aveq$ represents the average queue value, and q is the size of the instantaneous queue. The gain, w_q, determines the time constant of this low pass filter.

As long as the average queue size remains below a "minimum threshold" (Bmin), all packets are let through and not dropped. When the average queue size grows such that it falls between Bmin and some "maximum threshold" (Bmax), packets are dropped at the gateway with a certain probability that is linearly correlated to the average queue size. This is illustrated in Figure 3-1, and the following equation.

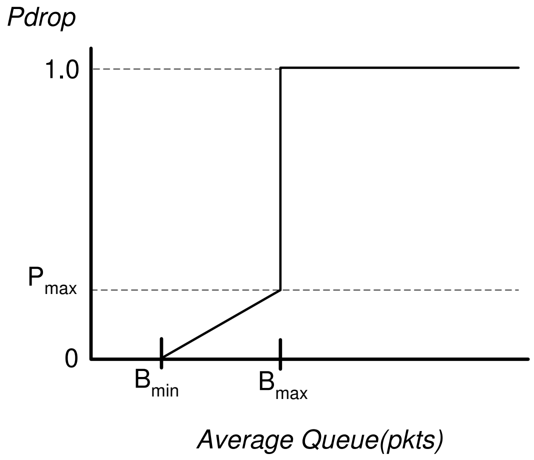


Figure 3-1: Original RED.

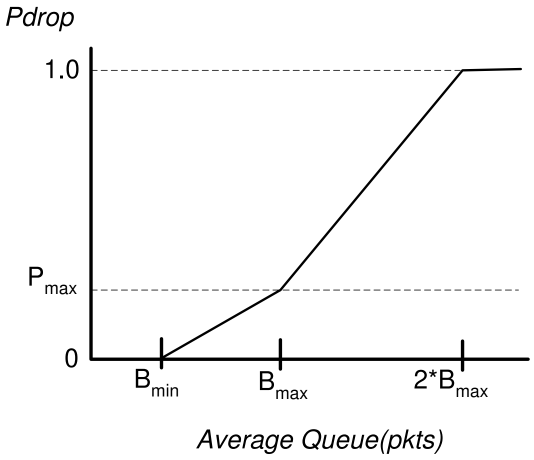


Figure 3-2: Gentle RED.

$$p_b \longleftarrow Pmax * \frac{aveq - Bmin}{Bmax - Bmin} \tag{3.2}$$

The effect of using the RED buffer management scheme is to keep average queue size low and overall throughput high, while reducing bias against bursty traffic [16].

Parameter Setting There are some guidelines for setting RED parameters [13], and most RED tests and simulations in the recent literature follow these parameter guidelines.

Pmax (max_p): The suggested setting for Pmax is 0.1, which represents a maximum drop rate of 10 percent. If the required steady-state drop rate exceeds this amount, we "assume that something is wrong in the engineering of the network, and that this is not the region that anyone wants to optimize." [13]

Bmax (max_{th}): It also suggests as a "rule of thumb" to set the maximum threshold Bmax to three times the minimum threshold Bmin. This was changed from two times Bmin in the 1993 RED paper. [16]

Bmin (min_{th}): Bmin must be big enough to allow a certain-sized burst to be absorbed.

Gain, or w_q: The gain, w_q, determines the time constant of the filter, measured in packet arrivals. The time constant is calculated by the formula: $-1/ln(1-w_q)$ [16]. If the instantaneous queue changes from one value to another, this formula indicates the number of packet arrivals it takes for the average queue to move 63 percent of the way from the first value to the second. If the time constant is too small, transients will be captured in the averaged queue value. If the time constant is too long, the response of the average queue will be too slow, and will become a poor reflector of the current queue level. Neither are desirable.

The appropriate setting of w_q is dependent upon the link capacity according to the following equation,

$$w_q = 1 - exp(-1/(T * C)) \tag{3.3}$$

where T is the time constant in seconds, and C is the link capacity in packets per second. RED achieves highest performance when the time constant is some multiple of round-trips, rather than a fraction of a single round-trip time [15].

3.1.1 ECN

The performance of RED can be enhanced by setting a special bit in the Internet Protocol (IP) header, called the Explicit Congestion Notification (ECN) bit [11]. Upon the detection of incipient congestion, the RED gateway can turn on this ECN bit in the packet header rather than drop the packet. This packet is said to be "marked". When the marked packet arrives at its destination, the ECN bit is copied into the packet's acknowledgment that is sent back to the sender. Once the TCP sender receives the acknowledgement and detects that the ECN bit has been set, it can respond to the congestion notification as though it had just detected a dropped packet via duplicate acknowledgments. Thus, ECN allows congestion to be indicated to the end-host without dropping packets. Using ECN in this manner supposedly has the benefit of avoiding unnecessary packet drops and delays. This also allows TCP to retransmit and recover losses sooner, rather than rely on a costly coarse-grained timer. [11, 1, 14] (See Chapter 2 for further background on TCP.)

In order for this scheme to work, not only do the gateways need to be modified, but TCP senders need to be ECN-enabled, so that they know to check the bit. Most of the simulations conducted in this paper turn on ECN. Note that wherever the term "dropping" is used within this paper, "marking" is also implied.

3.1.2 Gentle RED

The "gentle" version of RED was introduced to alleviate parameter sensitivity. It increases the packet drop rate gradually from Pmax to 1 as the average queue size grows from Bmax to 2 * Bmax [12]. Figures 3-1 and 3-2 show the difference between original RED and RED with the "gentle" addition.

Simulation tests show that even with less optimal settings of RED parameters Bmax and Pmax, gentle RED can still provide robust performance [10], and is with high confidence superior to the original RED. Zhang and Qiu [33] claim that gentle RED performs well in heterogeneous environments involving FTP, Web, and UDP. Therefore, all RED or RED-related simulations conducted in this thesis use the "gentle" form of the algorithm.

3.2 Weaknesses of RED

Much experience has already revealed the difficulty in parameterizing RED queues to perform well under different levels of congestion and with different traffic mixes.

Many researchers have advocated not using RED [6, 24, 23] for various reasons. One of these reasons is that the tuning of RED parameters has been an inexact science, with some researchers resorting to trial-and-error as a means to find optimal settings. In addition, numerous RED variants have been proposed, some of which are discussed below. Many researchers have turned to mathematical analyses and have begun advocating a self-tuning, control-theoretic approach to RED [18, 7, 20]. The fact that these require no manual tuning relieves much of the burden in setting RED parameters accurately.

Some of the complexity in classic RED can be attributed to its parameter sensitivity. RED is sensitive to the values of both w_q and Pmax, and therefore these must be carefully tuned in order to achieve good throughput and reasonable delays [15]. The gentle version of the algorithm improves its robustness in this regard, in that increased traffic loads can be controlled by the more steeply increasing drop rate, even when the average queue exceeds

Bmax.

In classic RED, the link utilization and queue length can vary over a range of Pmax and for various values of N [15, Page 4]. Floyd acknowledges that the choice of parameters embodies "conscious design decisions" and that the optimal values are not known, and further that these optimal values "would depend on a wide range of factors including not only the link speed and propagation delay, but the traffic characteristics, level of statistical multiplexing, and such, for the traffic on that link. These decisions about average and maximum queue size are embodied not only in the choice of buffer size, but also in the choice for min_{th} and w_q." [13]

If RED were ever to be widely deployed, it is highly important that RED be robust, and that there be a systematic and straightforward way of setting parameters.

3.3 Related Work

It has become widely recognized that in order for RED to be robust, it needs to be adaptive. In order for a congestion control algorithm to be robust, it must be able to handle varying traffic loads. Furthermore, RED's static parameters (or, at least, parameters which must be reset manually) limit the range of traffic loads that it can handle.

Many new adaptive algorithms have been proposed in the past few years. There is yet to be a document that clearly classifies and benchmarks them all. This is not that document. However, we describe in some detail several of the relevant works to provide the reader with perspective. Some of these algorithms share common points of emphasis with one another and with the 2RegionRED algorithm we propose in Chapter 4. Their emphasis lies in one or more of the following:

- Adaptation to changing loads
- Decoupling of the congestion measure from the queue size
- Use of the number of flows as a measure of congestion
- Throughput/latency tradeoff as a way to determine a proper queue operating point

3.3.1 Adaptive to Varying Traffic Loads: Variations on RED

The following algorithms are similar to RED in their mechanisms, but with parameters that adapt to changing environments. The common RED mechanisms are:

- Queue estimation using EWMA averaging
- Probabilistic dropping probability
- The idea of keeping the queue within thresholds

In these algorithms, as in classic RED, queue length remains the indicator of the level of congestion or load.

Adaptive RED was originally proposed in a paper by Feng, Kandlur, Saha, and Shin [7], and adjusts Pmax in order to keep the average queue length between Bmin and Bmax. An improvement on the original implementation of Adaptive RED was proposed by Floyd, Gummadi, and Shenker which automatically sets multiple parameters [15]. Pmax is adapted in response to measured queue lengths to keep the averaged queue in a target range spanning a region half way between Bmin and Bmax. w_q, the gain used in the queue estimator, is automatically set based upon the link speed. Robustness of the algorithm is attributed to a very slow and gradual adaptation of Pmax using AIMD (Additive Increase Multiplicative Decrease). Simulations involving Adaptive RED congestion control can be found in Chapter 6.

Load Adaptive RED is another approach [3]. It leaves Pmax fixed, but dynamically adjusts the Bmin and Bmax thresholds as the system load changes. The proposed mechanism is to adjust the thresholds in such a way that the packet loss rate is kept close to a pre-specified target loss rate. This target loss rate is chosen as one that would prevent excessive timeouts among the TCP connections. In order to achieve the target loss rate, the algorithm tries to maintain a constant packet dropping probability p_b over time, according to Equation 3.2. It notes loosely that as the number of flows vary causing the traffic load to increase or decrease, the average queue size *aveq* also increases or decreases respectively. In order to maintain a constant p_b, as $aveq$ changes, the thresholds can be adjusted accordingly, regardless of the setting of Pmax.

3.3.2 Decoupling the Queue Length

The algorithms in the previous section use queue length as a congestion measure. A growing number of researchers have, however, become convinced that queue length alone is not a practical measure of congestion.

BLUE [8] emphasizes that decoupling the queue length from AQM schemes can provide significant performance improvements. BLUE, evidently named because it is "a fundamentally different active queue management algorithm" from RED, uses buffer overflow and link idle events to adjust the packet marking/dropping probability for managing congestion. Upon sustained buffer overflow (which results in extensive packet loss), BLUE increments the marking probability by some factor. If on the other hand the link is idle, indicated by an empty queue, BLUE infers that its present dropping probability is too high, and so it decrements its marking probability.

REM or Random Exponential Marking [2] decouples what it calls the "congestion measure" from "performance measures". A REM link computes some measure of congestion by measuring both the queue size ("buffer backlog") and the difference between the aggregate input rate and the link capacity. Common performance measures include loss rate, queue length, and delay. In REM, each link marks with a probability that is exponentially increasing in the congestion measure. REM stresses that this exponential relation is important when multi-bottleneck paths are considered, because of its effect on the end-to-end marking probability, or what it calls the "path congestion measure". The endhost is able to infer the path congestion measure from the fraction of marked packets it receives in a time interval. With knowledge of the path congestion measure, the source then adjusts its rate using some chosen marginal utility relation. Decoupling also allows flexibility in that the same exponential marking mechanisms can be used with different congestion measures.

3.3.3 Relation to the Number of Flows

Load-adaptive RED [3] points out that the number of flows affects congestion levels and that this is reflected in the size of the average queue. The Adaptive RED algorithm also uses increasing queue size as a signal to increase its drop rate. However, the linear dropping probability curve that characterizes these variants of RED fails to capture the dynamics between the drop rate, number of flows, and the queue size.

Some algorithms therefore try to estimate the number of competing TCP flows N in a more direct manner. The drop rate is then adjusted accordingly. For these algorithms, the number of competing TCP connections is used as the measure of load or congestion.

FPQ or Flow-Proportional Queuing [27] varies the router queue length proportionally to the number of active connections. It estimates the number of active TCP connections using an algorithm that hashes connection identifiers of arriving packets into a small bit vector that is incrementally cleared. Once it has estimated N, FPQ determines what the target buffer size q_t should be as a function of N using the following function. P, the pipesize, is the delay bandwidth product of the link.

$$targetQueue(N): \quad q_t \longleftarrow max\left(\frac{P}{2N-1}, 6N\right) \tag{3.4}$$

One consideration for the target queue is that for small N, the target buffer must be large enough so that full link utilization can be sustained. Notice for the target queue that as N initially increases, q_t decreases linearly according to $P/(2N-1)$. 6N is required at minimum because six packet buffers are required per connection in order to avoid timeouts. Thus, when N increases past a certain point, the queue length actually grows. [2]

Once the target queue is computed, FPQ calculates the target loss rate. This is the dropping probability that hopefully will drive the queue length to the target buffer size.

$$targetLoss(q_t, N): \quad min\left[\left(\frac{0.87}{\frac{q_t+P}{N}+1}\right)^2, 0.021\right] \tag{3.5}$$

[2] Keep in mind that TCP Tahoe is used in the FPQ analysis, as opposed to TCP Reno.

Notice that the loss rate is proportional to N^2. The loss rate is chosen as that which would cause each of the N windows to be of size $(P + q_t)/N$. This relies on the result that when a particular loss rate p is experienced by steady-state TCPs over a long period of time, the resulting average window size is inversely proportional to the square root of p [22].

$$W \sim p^{-1/2} \tag{3.6}$$

The loss rate is limited to a maximum of 0.021 or 2.1% loss rate because anything greater proved to have little impact on performance. By keeping the queue large enough so that each connection has a window of at least six packets on average, FPQ is able to avoid most timeouts, resulting in a more predictable queuing delay.

FPQ-RED FPQ can use the RED queue management scheme "to enforce its target queue length and drop rate because of RED's ability to avoid oscillation" [26, Page 51]. When N increases to the point where the target loss rate is limited to 2.1% and the target queue grows proportionally to 6N, FPQ with RED can be thought of as a RED scheme that keeps Pmax fixed, but varies Bmax [27]. In this sense, FPQ and Load-Adaptive RED are similar in nature.

When FPQ is used with RED, Morris sets Pmax to the $targetLoss(q_t, N)$, Bmin to 5 packets, and Bmax to $2 * targetQueue(N)$ [26, Page 47]. A few possible troublespots when using FPQ with RED are also mentioned. For instance, RED's linear queue length to drop rate function doesn't quite match that which FPQ needs. Also, the setting of the gain for the EWMA averaging of the queue length may require more damping [26, Page 47]. Nevertheless, some simulations of FPQ with RED are presented in Chapter 6 to illustrate the algorithm's main principles. In this thesis, we will refer to this variation on FPQ as FPQ-RED.

SRED or Stabilized Random Early Drop [28] is another algorithm that measures load by estimating the number of active flows, though in a different manner. SRED uses a "zombie list" to estimate the number of flows. This is described in detail in the paper by Ott, Lakshman, and Wong [28]. Like FPQ, SRED derives some target buffer size based upon this knowledge of N and the average window size, and then computes a drop rate, proportional to N^2, whose aim is to drive the queue length to this target. The equations that SRED uses are more loosely derived but similar to those used in FPQ. Establishing the dropping probability only on the estimate of N can lead to unpredictable buffer occupancy, because the $p \sim N^2$ relation involves assumptions that do not account for flow start-up behaviors, varying RTTs and packet sizes, among other things. Thus, the final drop rate imposed by SRED is also influenced by the actual queue size. SRED differs from FPQ in that SRED assumes a fixed buffer size.

3.3.4 Tradeoff between Throughput and Delay

RED-light is the congestion control algorithm proposed in the 1999 draft titled "RED in a Different Light" [20].

RED-light notes that as the number of flows increases, the average queue size increases linearly while the drop rate increases in a quadratic relation. In an ideal world, one could infer N from the average queue size and then administer a drop rate proportional to N^2. Because real world traffic is much more complex, RED-light focuses on finding an appropriate operating point. [3]

The authors of RED-light suggest setting Bmin, the target buffer size, to 0.3 * P where P is the pipesize, or delay bandwidth product. They claim that this operating point represents:

1. a good balance between delay and utilization
2. a value that is able to absorb maximum bursts (bursts are also affected by the gain setting, which is described in detail in their paper) [4].

Between a fixed Bmin and Bmax, the drop rate increases in a quadratic manner with the queue size, to capture the relation between N, the size of the queue, and the drop rate. Note that there are some subtleties in the actual implementation of RED-light, which uses mechanisms that at least in appearance are radically different from RED. Also, a more deterministic method of dropping is used, rather than probabilistic.

[3] Caveat: Because the paper on RED-light [20] is a draft, some of these ideas are not completely articulated, and what is summarized here is the author's interpretation.

[4] The paper proposes a new queue estimation scheme to track the *persistent queue*. What the appropriate queue estimation scheme should be is explored in Chapter 5

Chapter 4

2RegionRED, the Algorithm

In the previous chapter, we have seen that the task of setting up a RED gateway is made complex by the number of parameters involved and the need to understand how these parameters are related to one another. We have also seen that it is difficult to parameterize RED to work effectively over varying levels of congestion. A workable RED would thus require maintenance in the setting and resetting of its parameters. The degree of human intervention that this involves is a clear invitation for human error to leak into the system, which is undesirable for the stability and clean operation of the Internet. For these reasons, it is of interest that a congestion control algorithm be:

- Simple. There should be an intuitively clear and systematic method of setting parameters (if any).
- Adaptive. The algorithm should be self-adjusting, requiring little or no manual tuning.

In this chapter we investigate a new adaptive RED algorithm, 2RegionRED. What distinguishes this algorithm from the others is a set of principles that guide its design. We provide a clear and intuitive way of thinking about how these principles can be used to determine the drop rate that is appropriate at any time [1].

These principles can be characterized by:

[1] This chapter summarizes and adapts ideas originally proposed in a 2000 draft by D. D. Clark titled "Slope-Controlled RED"

- Goals to maintain high utilization while keeping the persistent queue at a minimum
- A correlation between the number of flows, optimal buffer size, and drop rate
- Decoupling of the drop rate from the size of the queue

While these ideas are not new in themselves, we tie them together in a way that leads to a novel way of thinking about RED. The 2RegionRED algorithm is one product of this analysis.

Note that this section of the thesis attempts to work within the framework of the current Internet, leaving its protocols, structure, and state more or less intact. It leaves TCP unchanged, and focuses on the congestion control algorithm at the gateway.

4.1 Network Theory

4.1.1 An optimal RED signal: High link utilization, Low queuing delay

One performance metric that most users care about is goodput, or the effective throughput that each user can attain. The congestion control algorithm operating at a gateway can have a significant effect upon user goodput. Its role in maximizing goodput is twofold, and involves minimizing the latency incurred at a router while maintaining high link utilization.

Latency, or delay, is often used to quantify the quality of service delivered to a user, with better service characterized by less delay. There are four sources of delay that contribute to a user's perceived round-trip time. Propagation delay, the time it takes for a signal to propagate from one end of a link to another, is a result of the limitation imposed by the speed of light, and is therefore based upon the distance between source and destination, as well as physical factors such as the medium through which the bits are flowing. Serialization delay represents the time it takes to place a packet on the network. It is dependent upon the circuit speed; the higher the circuit speed, the less serialization delay. Packetization delay, the time it takes to collect enough bytes to fill a packet before it can be sent out, also contributes to the serialization delay. Finally, queuing delay is incurred when a packet has to wait in some buffer along its path because there are other packets ahead of it in the queue waiting to be processed. Usually, propagation and queuing delay are the dominant contributors to latency. Whereas physical laws prevent us from minimizing propagation delay, congestion control algorithms can indirectly control the amount of queuing delay a user experiences by carefully controlling the drop rate.

In addition to minimizing queuing delay, maintaining high link utilization is also important, because sustained periods of idleness on the link indicate that bandwidth is being wasted. This is wasted bandwidth that could have been used to increase connection throughputs. The congestion control algorithm should thus try to keep the link busy at all times.

These two goals of reducing the latency caused by queuing while keeping the link fully utilized translates into an optimal RED signal that keeps the buffer at a minimum, yet filled to the point such that it never goes to zero. We will revisit this important point in the next section.

4.1.2 The new RED signal: a more realistic RED signal

The optimal RED signal described above requires the algorithm to be omniscient, and is therefore not practical to implement. In this section we explain the *new RED signal* that is used in the design of 2RegionRED. This new RED signal tries to ensure full link utilization at the expense of some reasonable amount of queuing delay.

The role of buffers in routers is to absorb short time-frame bursts, as well as to buffer the packets that result from the congestion avoidance behavior of TCP, which involves the sawtooth in the window size as seen in Section 2.2.2.

As the various packet flows go through cycles of window increase and decrease, the aggregate occupancy of the buffer will go up and down, a phenomenon here called *congestion related buffer fluctuation*, or just fluctuation (in contrast to fluctuation caused by transients such as bursts). While in real networks the pattern of buffer fluctuation will be very complex, in a simplified analysis there is a well defined fluctuation behavior that will result depending on how congestion is signaled.

Figure 4-1 depicts the size of the congestion window for three individual TCP flows whose paths share a common outgoing link interface, and whose connection properties (RTT, bottleneck, etc.) are identical. Their aggregate window, which is simply a sum of the individual windows per unit time, is also depicted. The aggregate represents the number of packets that arrive at this link interface each round-trip time. Q represents the queue depth, or allocated buffer size. P represents the pipe size, which is the equivalent of the link capacity, or delay-bandwidth product for this link. If the incoming traffic exceeds the link capacity P, the excess is buffered and a queue begins to form. Thus, the shape of the queue roughly matches the shape of the aggregate window above P.

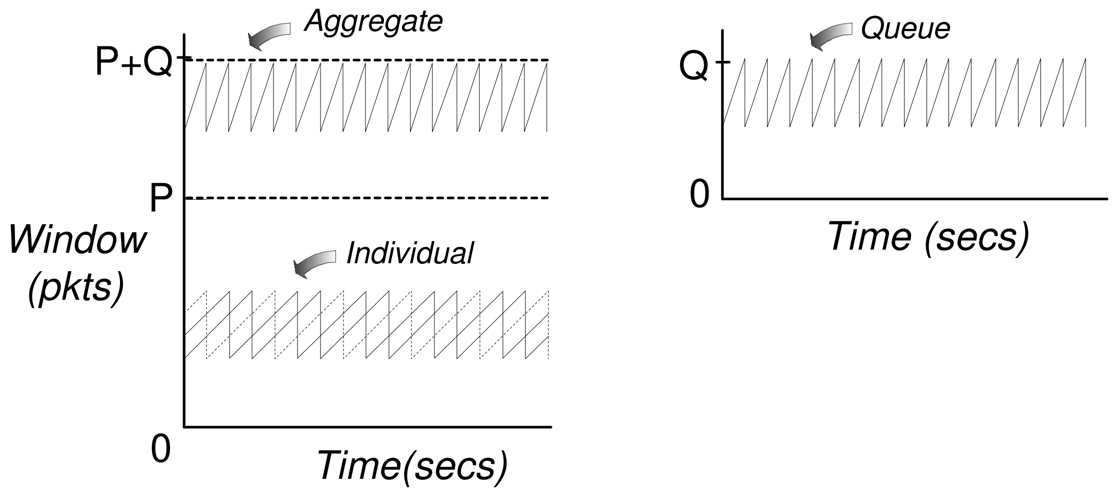


Figure 4-1: Windows of three TCP flows in congestion avoidance

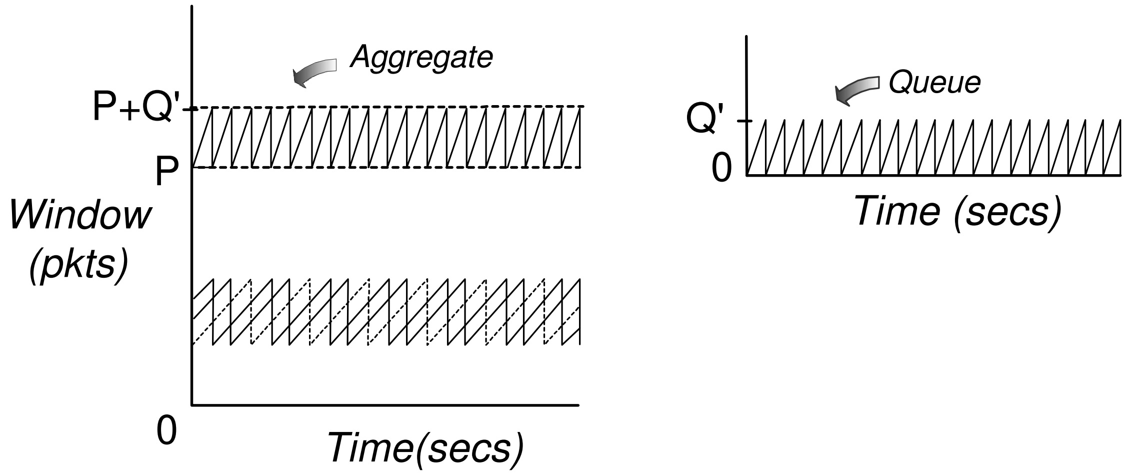


Figure 4-2: Desired dropping behavior for new RED signal

For this diagram, assume a very simplistic dropping mechanism, in which a single packet is dropped once the queue approaches its limit Q. In this ideal scenario, a packet is dropped from a different TCP each time, in perfect round robin order. We say that these flows are operating *in quadrature*, or *out of phase* with one another. Notice that the size of the fluctuations in the buffer approximates the change in the window size of a single TCP after experiencing a drop.

In the previous section, we noted that an optimal RED signal would keep the link just to the point of full utilization, but nowhere beyond that. Throughput would be maximized while incurring no cost in queuing delay, as a result of the absence of queues. Such optimal behavior would require the dropping algorithm to be omniscient. Having observed in Figure 4-1 the combined effect of TCP window dynamics with a simple router drop mechanism, we can imagine a more realistic RED signal that is motivated by the same principles behind the optimal RED signal. This is depicted in Figure 4-2, and it is this idea that forms the basis for the new RED algorithm presented in this paper.

In Figure 4-2, only one change has been made, which is the timing of the drops. The drops occur sooner. That is, they occur when the queue has reached just a fraction of the total queue size. Because it takes less time for the queue to grow to Q' than to grow to Q, the corresponding drop rate is higher, causing the drops to be spaced somewhat closer together. Different choices of Q' can thus be thought of as corresponding to different drop rates enforced by the router. Q' in this case was chosen to create the drop rate that would cause the buffer fluctuations to exhibit the following behavior. On each drop, the aggregate window falls to P exactly, a point where the link is running at its full capacity, but where there is no queue. In this way, the link is always running at 100 percent. Between drops, however, the queue grows such that connections experience some queuing delay. This buffer behavior will be referred to as the *desired behavior* at the queue, and the drop rate that leads to this behavior is referred to as the *required drop rate*. In general, this RED signal will be referred to as the *new RED signal*.

Notice that this new RED signal sustains full link utilization at the expense of some queuing delay. We call the level Q' the *optimal queue size* because it represents the minimum queue size required to maintain full link utilization.

4.1.3 Relating buffer size and behavior to the number of traffic flows

The left diagram in Figure 4-3 is adapted from "RED in a Different Light" [20]. It shows just the aggregate windows for systems with one, two, and three flows operating in steady-state. Of course, for the one flow system, the individual window and aggregate are the same. Notice in particular that as the level of multiplexing increases, the aggregate behavior of the buffer under RED changes. The frequency of drops increases and the size of the buffer fluctuations grows smaller. Here is the explanation. Because each TCP contributes a one packet increase in its window every RTT, the aggregate window reaches the queue limit sooner when there are more flows in the system. This results in the higher drop rate. The buffer fluctuations grow smaller for two reasons: when one flow cuts its window in half during a particular RTT, all the other N-1 flows are incrementing theirs by one thereby filling in some of the hole. Furthermore, because multiple TCPs must share the same bandwidth, the portion (window size) that each can attain decreases as the number of TCPs increase. This is significant because in a system where there are more competing flows, the queue reduction that results from a TCP's window being halved is smaller simply because the window sizes are smaller [20]. Therefore, if RED can keep the congestion avoidance behavior of the different flows in quadrature then the observed fluctuation will decrease in size as the number of flows sharing the same output link interface increases.

The corresponding new RED signal is shown in the righthand diagram of Figure 4-3. Because the buffer fluctuations grow smaller as more flows are added into the system, a higher degree of multiplexing actually leads to smaller queues, more closely approaching the optimal RED signal of Section 4.1.1.

Equations

We can characterize the key properties of the new RED signal with a couple of formulas. The following notation is used:

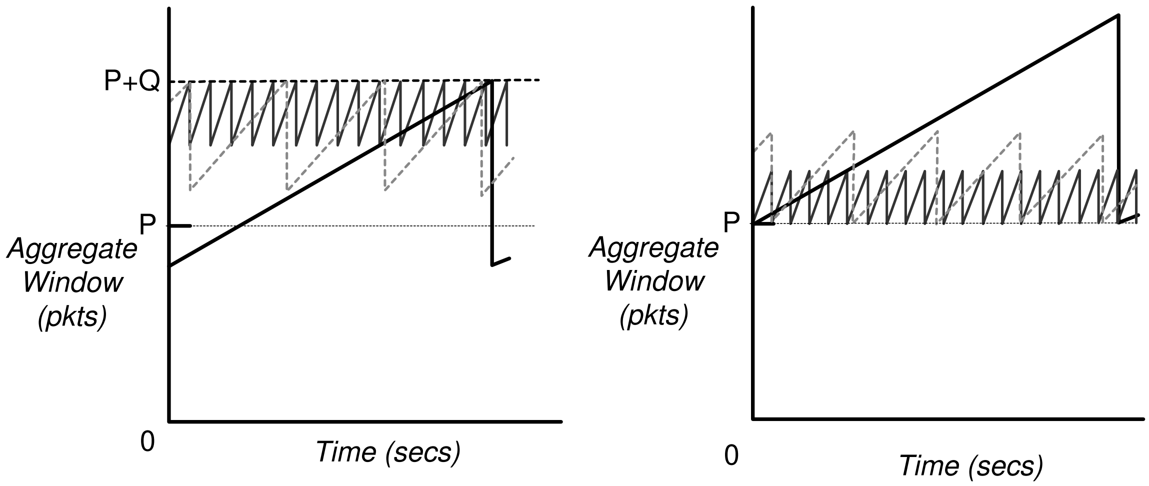


Figure 4-3: Effect of multiplexing on the aggregate window, (Left) Original, (Right) New RED signal

Let B = bandwidth, in pkts/sec

Let D = round trip, in secs

Let N = number of flows

In the following, we compute the expected fluctuation in buffer size (or the optimal queue size), and the corresponding drop rate as a function of the number of flows N, and the delay bandwidth product $D * B$. All of these calculations assume the perfect quadrature behavior of identical TCP flows.

Computing the optimal buffer size After experiencing a drop, a single TCP's window is cut from $2W$ to W, or a reduction of W. It is easy to approximate the optimal buffer size required for the new RED signal. In order to maximize the link utilization while reducing latency, the new RED signal is designed so that upon a drop, the buffer falls precisely to zero. Thus,

$$\textit{Optimal buffer size} \approx W \tag{4.1}$$

Assume that there are N flows, each with a minimum window size W. Because of TCP's AIMD congestion avoidance behavior, the window size will swing from W to $2W$, in the familiar sawtooth, which yields an average window size of $(3/2) * W$ per connection. The average number of packets in the system (in the pipe or in the buffers) at any time is the sum of the windows, or $(3/2) * N * W$.

$$Average\ Packets\ per\ RTT = (3/2) * N * W \tag{4.2}$$

If W is the size of the drop in the aggregate window for the new RED signal, as depicted in Figure 4-2, the average number of packets can also be represented as $Pipesize + W/2$, where $Pipesize = DB$. Equating this with Equation 4.2, we can compute a value for W.

$$\begin{aligned} (3/2)NW &= DB + (W/2) \\ W &= \frac{2DB}{3N-1} \end{aligned} \tag{4.3}$$

Without loss of generality, we use this simplification:

$$W = \frac{2DB}{3N} \tag{4.4}$$

The key observation is that as N increases, the optimal buffer size actually decreases.

Computing the drop rate The drop rate at the gateway is simply the inverse of the number of packets sent between drops. This can be visualized by the area beneath one sawtooth of the aggregate window curve. This drop rate is equivalent to that experienced by each individual flow, so it suffices to look at a single flow and compute the number of packets sent within its individual TCP sawtooth. Because of TCP's additive increase, it takes $W+1$ round trips for the window to grow from W to $2W$, averaging $(3/2)W$ packets per round trip. This computes to $(3/2)W(W+1)$ packets between drops, or roughly $(3/2)W^2$. Substituting the value for W from Equation 4.4, we arrive at the following:

$$\begin{aligned} Packets/drop &= (3/2)W(W+1) \\ &\approx (3/2)W^2 \end{aligned} \tag{4.5}$$

$$
\begin{aligned}
&= \frac{2}{3}\left(\frac{DB}{N}\right)^2 && (4.6)\\
Drops/packet &= \frac{3}{2}\left(\frac{N}{DB}\right)^2 && (4.7)
\end{aligned}
$$

The key observation is that as N increases, the desired drop rate increases according to a quadratic relation.

Computing the drops per round trip It is also insightful to compute the number of drops per round trip required of the new RED signal. We compute this using Equation 4.6 and approximating the average packets per RTT to be DB.

$$
\begin{aligned}
RTTs/drop &= (packets/drop)/(packets/RTT)\\
&= \frac{2}{3}\left(\frac{DB}{N}\right)^2 * \left(\frac{1}{DB+(W/2)}\right)\\
&\approx \frac{2}{3}\left(\frac{DB}{N}\right)^2 * \left(\frac{1}{DB}\right)\\
&= \frac{2DB}{3N^2} && (4.8)
\end{aligned}
$$

This number can either be much smaller or much bigger than one, which is part of the difficulty of tuning RED, as we will see shortly.

Numerical examples

It is helpful to look at actual numbers, rather than equations in a number of unknowns, because a sense of the actual magnitudes of values often helps. Table 4.1 shows among other things how the optimal buffer size and corresponding drop rates for a 1Gbps bottleneck vary with the number of flows N, as required under the new RED signal. The table assumes a round trip delay of 100ms and 1500-byte packets. The following list the notations and summary of the equations used to generate the table.

Let PPD = Packets Per Drop

Let DPP = Drops Per Packet

N	TPUT	PPD	DPP	RPD	DPR	W(Optimal Buffer)
1	1e+09	1.04e+08	9.6e-09	8.33e+03	0.00012	8.33e+03
2	5e+08	1.67e+07	6e-08	1.67e+03	0.0006	3.33e+03
5	2e+08	2.13e+06	4.7e-07	238	0.0042	1.19e+03
10	1e+08	4.96e+05	2.01e-06	57.6	0.0174	575
40	2.5e+07	2.96e+04	3.37e-05	3.53	0.284	140
75	1.33e+07	8.42e+03	0.000119	1.01	0.995	74.4
100	1e+07	4.74e+03	0.000211	0.567	1.76	55.7
200	5e+06	1.2e+03	0.000831	0.144	6.94	27.8
500	2e+06	202	0.00495	0.0242	41.3	11.1
1000	1e+06	54.7	0.0183	0.00656	152	5.56
1500	6.67e+05	26.1	0.0383	0.00314	319	3.7
2000	5e+05	15.7	0.0635	0.00189	529	2.78

Table 4.1: Choosing N_{Bmin}: 1 Gbps link, 100ms RTT. $TPUT$ refers to the maximum bandwidth, in bits per second, that each of N flows could receive. W refers to the optimal buffer size (or the size of one TCP window reduction) in packets. The packet size assumed throughout this paper is 1500 bytes.

Let RPD = RTTs Per Drop

Let DPR = Drops Per RTT

$$\begin{aligned}
W &= \frac{2DB}{3N-1} \\
PPD &= (3/2) * W * (W+1) \\
DPP &= 1/PPD \\
RPD &= \frac{PPD}{D * B + (W/2)} \\
DPR &= 1/RPD
\end{aligned} \tag{4.9}$$

These numbers hint at some of the problems of setting up classic RED properly. For one flow attempting to fill the 1 Gbps link fully the required drop rate in drops per packet is $9.6 * 10^{-9}$, which is a number so small as to be hard to regulate. The interdrop interval is 833 seconds, or about 14 minutes! On the other hand, for 1000 flows, each of which can go a megabit (not an unreasonable operating point), the correct dropping algorithm must produce 152 drops per round trip. In other words, the drop rate can vary by 6 orders of magnitude.

Another interesting observation is that as the number of flows increases, the optimal queue size decreases. The necessary minimum buffer size to sustain full link utilization is around 8333 packets for one flow, 575 packets for 10 flows, and just six packets for 1000 flows. [2]

In summary, if we can develop a scheme that can dynamically estimate the number of flows and adjust the drop rate accordingly, then we will have an algorithm that self-adapts to varying levels of congestion. In a sense, the addition of the "gentle" parameter to classic RED extends the range of drop rates the algorithm can administer and hence the range of congestion levels that can be controlled [12]. However, the "gentle" addition can accomplish this only to a limited degree.

4.2 A Design Rationale: Two Operating Regions

As described in Chapter 3, the traditional RED algorithm operates by adjusting the dropping probability according to the current size of an EWMA-averaged queue. As the queue increases, traditional RED increases the dropping probability in order to control the queue size. However, as the above analysis for the new RED signal shows, queue size is a poor basis to set the dropping probability. This is because as the number of flows goes up, the required drop rate goes up, but the optimal queue size actually goes *down*. In other words, we would like smaller queues to be associated with a higher dropping probability that is needed to control a higher number of flows. However, if the dropping probability is to be tied to the buffer size, it is hard to set up a scheme without the buffer size growing as N increases. The reader is encouraged to try this mental exercise.

One approach to the design of a workable RED algorithm is to note that a successful scheme must span two operating regions, distinguished by the number of drops per RTT

[2] Note that there is a certain point where the number of flows grows so large that performance begins to degrade for other reasons. When the 1Gbps link is shared by over 1500 flows, the average W shrinks to less than four packets. This is not enough packets for a TCP to recover from multiple packet drops within a round trip without having to timeout. Timeouts are very costly in performance and throughput. This realm where the number of flows sharing a particular bandwidth becomes suffocating is outside the scope of this paper. It is addressed in other sources [27].

required to maintain the goals of high utilization and low buffer occupancy described in Section 4.1.1. The scheme assumes that the buffer's averaging algorithm smoothes over some reasonable span, on the order of a small number of round trips. Smoothing over anything less than a single round-trip time will allow sub-round-trip fluctuations to distort the queue estimation.

High-N Region First consider a bottleneck handling a large number of flows. In the 1 Gbps example in Table 4.1, 1000 flows corresponds to a drop rate of 152 drops per round trip. Suppose the algorithm drops packets at exactly this rate. Because there are many drops in one round-trip, the fluctuations caused by the congestion adaptation will be masked, or smoothed out by the queue averaging mechanism along with any sub-round-trip fluctuations. The result will be a more or less steady-state buffer estimation. In this scenario, a reasonable goal for RED is to hunt for the drop rate that keeps the buffer size more or less constant. If the algorithm can hone in on the correct drop rate, the resulting behavior should approximate that of the new RED signal. If N is the number of flows, we call this region the High-N Region because a large number of flows must be steadied by a high drop rate, $P_{drop} >$ *1 DPR (1 Drop per RTT).*

Low-N Region Now consider a bottleneck through which a small number of flows traverse. In the 1 Gbps example, 10 flows require roughly 58 round trips to elapse before a drop. Because 58 round trips is long in comparison to the time constant used by the queue averaging algorithm, the steady growth in the queue during these round trips will not be smoothed over. As a result, the sawtooth of the aggregate window and corresponding queue will become "visible" to the algorithm. The goal for RED in this region is to take advantage of some aspect of the sawtooth to determine the appropriate level of dropping, in accordance with the new RED signal. In the following section, we see how the slope of the sawtooth curve might be used to determine the proper drop rate. We will refer to this curve that tracks the growth in the buffer occupancy as the *buffer occupancy curve.* We call this operating region the Low-N Region, because it is characterized by requiring $P_{drop} \leq 1DPR$ (or $\geq 1RTT$ *per drop*), resulting from the small number of flows.

B	B(pkts/sec)	D(sec)	$\sim N_{1DPR}$
1 Tbps	83333333	0.1	2357.0
100 Gbps	8333333	0.1	745.4
10 Gbps	833333	0.1	235.7
1 Gbps	83333	0.1	74.5
155 Mbps	12916	0.1	29.3
60 Mbps	5000	0.1	18.3
10 Mbps	833	0.1	7.5

Table 4.2: N_{1DPR} values for varying bandwidth, 100ms RTT

Computing $\mathbf{N_{1DPR}}$ The behavior of the algorithm in the High-N and Low-N regions is directly related to the timescale over which the queue estimation algorithm smoothes. Whether the algorithm should behave according to the Low-N or High-N region is dependent on whether the required drops per round-trip is greater or less than one. We can calculate the number of flows N_{1DPR} for which a drop rate of one drop per round trip is required by determining when DPR = 1.0.

$$\begin{aligned}
DPR &= 1 \\
\frac{DB}{PPD} &= 1 \\
\frac{DB}{(3/2)W^2} &= 1 \\
\frac{DB}{\left(\frac{3}{2}\right)\left(\frac{2DB}{3N}\right)^2} &= 1 \\
\frac{N^2}{(2/3)DB} &= 1 \\
N &= \sqrt{(2/3)DB} \approx N_{1DPR} \qquad (4.10)
\end{aligned}$$

4.2.1 Slope Controlled RED for the Low-N Region: A Preliminary Proposition

In this section we entertain the notion of using the slope of the changing queue size to adjust the drop rate when the system is operating in the Low-N Region. The purpose of this section is to demonstrate how the two regions outlined above call for different mechanisms if adjustments to the drop rate are based upon some aspect of queue size.

The High-N Region uses past history spanning many drops to hone in to the correct drop level. This mechanism is suitable for that region because multiple drops occur in a single round-trip, allowing the algorithm to more quickly hone in. In the Low-N region, however, many round trips may pass between drops. In the 1Gbps example, for 10 flows the desired drop rate is one drop per 5.7 seconds (assuming a 100ms RTT). That means that the algorithm has to wait several seconds to see the effect of a single drop and to make a single adjustment. As a result, if the same mechanism used in the High-N Region is used here, the overall rate of adaptation may be very low. Moreover, if it takes so long to make a single adjustment, the algorithm will not be responsive enough to changes in the traffic that occur on a smaller timescale.

Slope as an indicator of N Therefore, the Low-N Region needs to adopt a different mechanism, one where the algorithm can adjust its estimated drop rate many times between drops. This requires that the algorithm look at some aspect of the queue size that has short-term visibility. In this region where the queue averaging mechanism cannot mask the shape of the sawtooth, the desired drop rate may be computed from the shape of this curve sampled at different intervals. One obvious characteristic of the buffer occupancy curve is its slope, measured in packets/RTT. If 10 flows are running in congestion avoidance, and share the same round-trip, every round-trip time the queue should increase by 10 extra packets. Therefore, if the algorithm measures that the queue rises by N packets within a round-trip time, it estimates that there are N flows. The slope of the changing queue size is a direct indication of the number of flows N, and thus a direct indication of the necessary drop rate. We call this idea *Slope Controlled RED for the Low-N Region*.

In the following, we identify a relationship between slope, round-trip delay, and the drop rate.

Let D = round-trip delay (in seconds)

Let N = number of flows

Let S = slope = N/D packets/s.

To compute the number of drops per second, divide the capacity D*B of the link (in packets per second) by the packets per drop. The number of packets per drop is simply 3/2*W*W packets, as computed in Equation 4.5.

$$\begin{aligned} Drop/s &= \frac{DB}{(3/2)W^2} \\ &= \frac{DB}{\left(\frac{3}{2}\right)\left(\frac{2DB}{3N}\right)^2} \\ &= \frac{3N^2}{DB} \end{aligned}$$

Note that N/D is equivalent to the slope of the buffer occupancy curve, so this reduces to:

$$Drop/s = \frac{3NS}{B} \tag{4.11}$$

The router has knowledge of B and S. Unfortunately it has no knowledge of N. Because of the relationship between S, N, and D, Equation 4.11 reveals that the router would either need to know one of N or D in order to work. Without this information, it might simply assume some canonical RTT value to represent a "typical" round-trip delay value.

Difficulties in Slope-Controlled RED

In simulation, we find that the slope is not a reliable indicator of the number of flows, even if D, the average RTT, is known. This is largely because the assumptions on which the new RED signal is based are threatened by the nature of real Internet traffic, which involve phenomena such as the exponential growth in queue size during slow starts, large transient bursts aggravated by ACK-compression, etc. This is discussed further in Section 4.3.3.

Another complication arises in Slope-Controlled RED in discerning when the algorithm is transitioning between the Low-N and High-N regions. Assume that the High-N region operates generally as described earlier in this section, adjusting its drop rate in effort to keep the buffer size constant. If we begin in the Low-N region, and the number of flows increases, at some point roughly around $N = N_{1DPR}$, the algorithm will want to switch over to the High-N mechanism whose goal is to keep the slope of the averaged queue size at zero. Switching back from the High-N to the Low-N region also encounters this same complexity. It is difficult to achieve accuracy in this "middle" region.

Without any explicit knowledge of N, finding a scheme that can produce the behavior of the new RED signal is not as easy as we thought.

4.3 2RegionRED

If we back away from the goal of absolute minimum buffer occupancy, it may be possible to define a variant of RED that is similar in principle to the original proposal, but represents a reasonable compromise in buffer behavior.

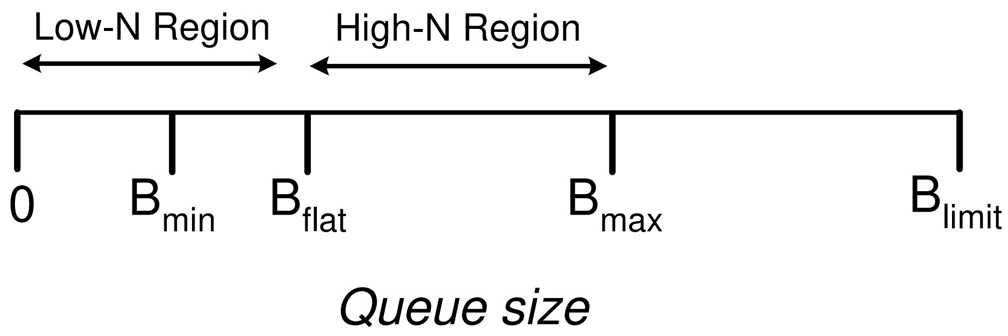


Figure 4-4: 2RegionRED: Two regions of operation.

We propose a variant of RED called 2RegionRED. The goal of this algorithm is to keep the average buffer occupancy controlled around or below a target value Bflat. 2RegionRED is so named because it attempts to distinguish the two operating regions as described in Section 4.2, and adapt its dropping mechanism accordingly. The actions taken by 2RegionRED can be classified into four phases as follows, depending on the size of the average queue. See Figure 4-4.

(I) Average buffer size less than Bmin: no dropping.

(II) Average buffer size exceeds Bmin but less than Bflat: drop one packet, and then refrain from dropping another for roughly 1.5 to 2 round trips so long as the buffer remains less than Bflat. This is the Low-N Region.

(III) Buffer between Bflat and Bmax: dropping probability adjusts dynamically in a way that is decoupled from the size of the queue. This is the High-N Region.

(IV) Buffer beyond Bmax: Either drop with some high probability such as 10% that is sufficient to handle worst-case "problem nets". Or, apply the "gentle" parameter as used in RED (See Section 3.1.2) to increase the drop rate linearly from some value to 100% as the queue grows from Bmax to 2*Bmax.

4.3.1 Low-N Region

Phase I and Phase II fall in the Low-N region of the algorithm, where the drop rate necessary to control the behavior of the queue is less than one drop per round-trip (equivalent to having multiple round trips between drops). According to this part of the algorithm, a single packet is dropped whenever the queue crosses the Bmin threshold. Below Bmin, the algorithm remains silent and no dropping ever occurs. There is also a small "hold-off period" when the queue is between Bmin and Bflat where no dropping is allowed. This is discussed later.

If things work properly, after the single drop at Bmin the buffer will fall below Bmin. As TCPs in congestion avoidance increase their windows once per round-trip, the buffer will continue to rise steadily. The time it takes for the buffer to refill to Bmin is thus a simple function of the average round-trip and the number of flows. Note that when the algorithm is operating in this Low-N Region, the regularity of the drops can be used to calculate the effective drop rate. This implicit drop rate is dependent upon the position of Bmin, the round-trip and number of flows.

The deterministic dropping at Bmin should bear resemblance to the regular dropping used in the analysis to compute the new RED signal. We will see how the new RED signal ties into the setting of Bmin.

In the remainder of this paper, the variable N_{Bmin} will be used to refer to the number of flows that is used to set Bmin. The variable N is used to refer to the actual number of flows sharing the bottleneck, and can vary from time to time. Thus, Low-N and High-N refer to regions responsible for handling small and large numbers of flows respectively.

Setting Bmin

The choice of Bmin is partly a policy decision, and partly a performance decision. For 2RegionRED, Bmin should be set to the maximum of

(a) The size that permits a TCP to attain some maximum desired throughput rate. The implicit drop rate that corresponds to this setting must be smaller than one drop per round-trip.

(b) The size that provides reasonable protection against idle periods due to swings in offered traffic and congestion regulation.

Only experience with offered traffic and the capabilities of the algorithm can confirm if condition (b) will be met. In the following, we explain the setting of Bmin that is necessary to satisfy condition (a).

Continuing with the 1 Gbps example, a network administrator deploying 2RegionRED decides that she wants to limit the maximum throughput for any one TCP to be one tenth of the link capacity, or 100 Mbps. This would allow a maximum of N=10 flows to each achieve a maximum of 100 Mbps. This does not seem to be an unreasonable proposition for a commercial network. In the following we explain why condition (a) in this scenario requires that Bmin be at least 575 packets.

575 packets represents the size of the optimal buffer required for 10 flows, as computed by Equation 4.3 according to the new RED signal. This information is displayed in Table 4.1 for a range of N. A round trip D of 100ms and packet size of 1500 bytes is assumed. For a bottleneck through which 10 flows are running in steady state, a single packet drop will cause the queue to drop by 575 packets on average. Ideally, if a single packet drop occurs at Bmin=575 packets as Phase II dictates, this will cause the queue to drop precisely to zero, minimizing queuing delay while maintaining full link utilization. The behavior of the

algorithm in this region is no different from that used to compute the new RED signal (for 10 flows). Because full link utilization is sustained, each of the ten flows is able to maintain the maximum desired throughput rate of 100Mbps. To summarize,

Let B = Bottleneck bandwidth
Let B' = Maximum throughput attainable by a single flow (B / N_{Bmin}).

$$Bmin = \textit{Optimal buffer size (as computed in Equation 4.3 where } N = N_{Bmin}\textit{)} \tag{4.12}$$

In our example, B is 1Gbps, and N_{Bmin} is 10. The computed Bmin of 575 packets enables 10 flows to each attain throughputs of B' = 100Mbps. The implicit drop rate that is associated with this setting of Bmin also corresponds directly to that computed in Table 4.1. For 10 flows with a 100 millisecond round-trip time, the implicit drop rate is about 0.0174 drops per round-trip (or 58 round trips per drop). This falls below the one drop per round-trip boundary that the Low-N Region can control, which is a necessary condition for the setting of Bmin.

It is because it takes at least a round-trip between the time a drop is signaled and the time that any resulting reduction in the queue can be seen, that the Low-N Region can at most signal one drop per round-trip. Therefore, the drop rate enforced by the Low-N Region can control at most N_{1DPR} flows. In the boundary case of our example, one drop per round trip occurs when there are about N_{1DPR}=75 flows.

When N does not equal $\mathbf{N_{Bmin}}$ What happens when the number of flows N does not equal N_{Bmin}? We find that there is some compromise in the behavior of the algorithm. For any $N < N_{Bmin}$, there will be periods of underutilization. This is illustrated in Figure 4-5. Because there are fewer connections, each is able to grab more of the bandwidth, resulting in window sizes that are larger than if the same amount of bandwidth was shared by $N = N_{Bmin}$ flows. The drop at Bmin therefore causes the queue to drain for a little while longer leading to periods of link underutilization. Thus, another way to think about N_{Bmin} is that it represents the minimum number of flows needed to sustain full utilization of the link. Note that one disadvantage of this scheme is that there will be some cases when N is very small,

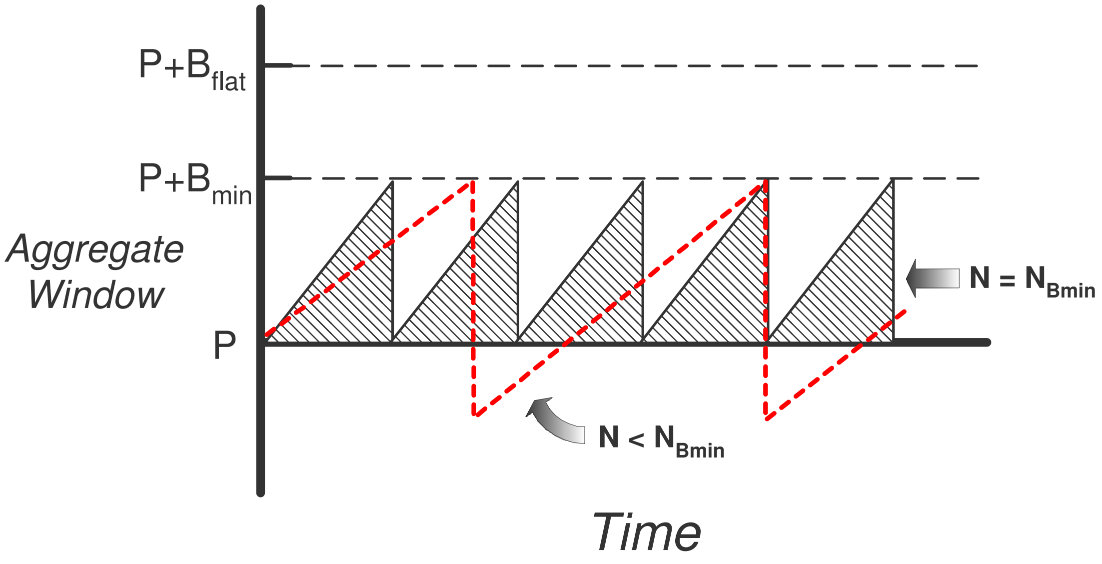


Figure 4-5: Low-N Region.

that full utilization of the link cannot be obtained.

When $N_{Bmin} < N \leq N_{1DPR}$, the cost is a higher queuing delay. Because there are more TCP's sharing the bottleneck bandwidth, their windows cannot grow as large. Because the windows are smaller, the queue reduction following a drop at Bmin will also be smaller. As a consequence, the link will be kept busy at all times, but the queue will never fully drain. Note that the higher N is, the smaller the fluctuations around Bmin. This results in a higher average queue, which leads to increased delay times. This is illustrated in Figure 4-6.

The analysis so far has assumed an ideal sort of orchestration where all N TCPs alternate dropping, and where the TCP windows all reach a specific size before they are halved upon receiving a drop signal. In reality, the chance of this happening is next to none. Among a multitude of other things, the size of successive window reductions will vary. If a TCP with a very small window receives a congestion signal at a router shared by many flows, it is very possible that the resulting buffer reduction will be unable to bring the queue back below Bmin. Or, if an ACK from reverse traffic is dropped, then the algorithm relies utterly on the High-N region to push the queue back into the Low-N region, if that is the appropriate region of operation [3].

[3]The complex dynamics of reverse traffic needs further study.

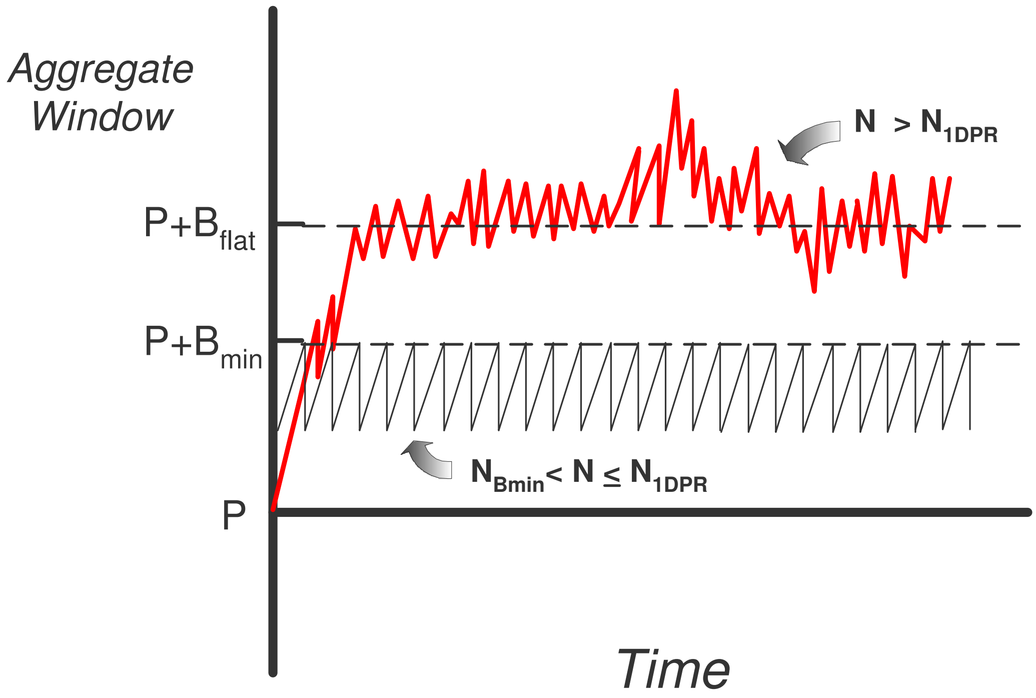


Figure 4-6: Approaching High-N Region.

In this case, the queue will continue to rise from Bmin to Bflat, in that hold-off period where no dropping occurs. As N approaches N_{1DPR}, the more difficult it is for the algorithm to keep the system operating in the Low-N Region. In Section 4.3.2, we will see how 2RegionRED handles these excursions into the High-N Region.

Note on Randomness In RED, the use of a probabilistic dropping function reduces the likelihood of synchronization among flows and ensures that the fraction of packets dropped by any single flow is proportional to that flow's bandwidth share. No particular flow is discriminated against.

In 2RegionRED's Low-N algorithm, a non-probabilistic dropping method is used. We claim that the chances that the deterministic drop at Bmin causes severe penalization to any one flow is extremely slim, and would require a very synchronized and unrealistic system.

Consider the 1Gbps bottleneck configured for $N_{Bmin} = 10$. If there are 10 flows, the system will experience a drop roughly once every 57 RTTs (See Table 4.1). This is approximately 500,000 packets between drops. In spite of there being so few flows, within 57 RTTs it is likely that the system will experience some transients and changes in the environment, such as the coming and going of flows. Furthermore, a longer span of time allows the per-connection clustering of TCP packets to be more spread out. This makes it unlikely that the same flow will be penalized each time the estimated queue crosses Bmin, assuming the flow even has that much data to send.

Setting Bflat

Once Bmin has been set, the setting of Bflat is straightforward. The region between Bmin and Bflat in Figure 4-4 is a silent hold-off period that exists to accommodate the fact that there is roughly a round-trip time between the time that a packet is dropped at a router, and the time that the TCP in question reacts and the router finally "sees" the consequence of its drop. During this round-trip, the average queue size will continue to grow to reflect the increasing windows of all the TCP connections (unless the environment changes, some connections terminate, etc.). The router refrains from dropping any packets during this interim period, because excess drops in this period are costly–they may lead to link underutilization. Rather than take any unnecessary actions, the router waits to see whether the drop it just administered is sufficient to control the queue size. By "controlling the queue size" in the Low-N Region, we mean bringing the queue back below Bmin after a drop.

The Bflat-Bmin region must therefore be big enough to accommodate the growth that is possible during this hold-off period. Because the Low-N Region can only control at most $N_{1DPR} = 75$ flows (in the Gbps example), the queue occupancy can at most grow by 75 packets during this waiting period. This suggests that Bflat should be set to Bmin + N_{1DPR}. This assumes, however, that all flows share the same round-trip, and are all operating in congestion avoidance. Because we do not know what the actual round trips are, and because some flows may be operating in slow start phase, we suggest setting Bflat a little larger than this. Several of the simulations in this paper set Bflat as follows.

$$Bflat = Bmin + 1.5 * N_{1DPR} \tag{4.13}$$

Notice how both the settings of Bmin and Bflat still rely on the elusive value of the average round-trip. We take the approach of assuming a *canonical RTT* value of 100ms, and analyze the algorithm's performance when the actual RTT is less than or greater than this value.

Simulations demonstrating the Low-N Region can be found in Section 6.2.

4.3.2 High-N Region

When the number of flows N sharing a bottleneck exceeds N_{1DPR}, the required drop rate exceeds 1 drop per round-trip. At this point, the average window size of the flows has become too small relative to the number of flows. The single drop that occurs at Bmin is no longer able to keep the queue operating in the Low-N Region. Above Bmin, the queue grows undeterred from Bmin to Bflat, that region that serves only as the hold off time for the Low-N algorithm. Once the queue exceeds Bflat, however, the algorithm enters the High-N Region where the drop rate increases to handle the increased number of flows.

Decoupling drop rate from the buffer size

The goal of the algorithm in the High-N Region is to adjust the drop rate to keep the queue around some target value Btarget. It is therefore obvious how, in the High-N Region, the algorithm deviates from the minimum buffer occupancy goals of the new RED signal. Rather than try to keep the queue around zero, 2RegionRED sets the target buffer somewhere above Bflat.

In essence, what 2RegionRED does is it adjusts the drop rate according to an implicit estimate of the number of TCP flows N. Many different schemes or variations of schemes can be used in this region.

In designing the algorithm to be used in the High-N Region, we must consider three things:

- when to make adjustments in measuring the average queue
- when to make adjustments in the drop rate
- how those adjustments are made

We propose the following algorithm for the High-N region.

When the queue first crosses the Bflat threshold, the drop rate P_{drop} is set to 1/DB which is roughly correlated to one drop per round-trip. This is a logical initial drop rate as it marks the boundary drop rate between the Low-N and High-N regions, allowing for a smooth transition between the two as the number of flows changes. This initial drop rate also handles the situation mentioned earlier when the queue operating in the Low-N region makes an excursion above Bflat. In such circumstances when there are few flows, this value of P_{drop} must be sufficient to knock an excursion above Bflat back into the Low-N Region's control.

If the number of flows is indeed greater than N_{1DPR}, then the queue will continue to grow above Bflat unless more aggressive dropping is used. We adjust the drop rate in the following manner.

Observing incremental changes in queue size Once above Bflat, adjustments are made to P_{drop} at regular intervals on the order of a (canonical) round-trip.

Section 1.2 described briefly the closed control loop interaction between the AQM scheme at the core and TCPs at the edge. It takes roughly a round-trip for the effect of drops administered at a router to impact the queue size.

The idea in the High-N Region is to infer the appropriate drop rate from looking at:

- dQ (delta-Q), the change observed in the estimated queue size over the last interval
- the drop rate administered in the interval before last

By observing the effect of the latter upon the former, we can infer how close the drop rate was to the desired drop rate (that which would have kept the queue steady), and then make a correction to the drop rate.

Ideally, the algorithm would apply the exact number of drops within an interval such that the queue size remained unchanged, or $dQ = 0$. This would require omniscience, however, with regard to varying round trips, packet sizes, flows entering and leaving or changing routes, among many other subtler interactions.

In reality, we expect the queue to grow and shrink. Let T1 and T2 refer to two intervals, where T1 immediately precedes T2. Assuming that the number of flows remains relatively steady, suppose that during T2 the queue grows by some amount dQ. What this implies is that the drop rate applied during T1 was too small. If the average size of a single window reduction is W, then the growth in the queue indicates that the number of packets the router dropped during T1 was roughly dQ/W short of ideal. The reverse applies if dQ is negative.

With this in mind, we arrive at a simple relationship at what the new Drops Per RTT (`newDPR`) should be by making an adjustment to the dropping probability at T1 (`origDPR`). This adjustment is referred to as the change in the DPR (`dDPR`) and from the reasoning above, is set to dQ/W.

$$\begin{aligned} newDPR &= origDPR + dDPR \\ &= origDPR + dQ/W \end{aligned}$$

The pseudocode for the High-N Region is presented in Figure 4-7. A canonical RTT of 100ms is assumed, and therefore this routine is called roughly once every 100ms. In the implementations, we include a random offset to this value to avoid synchronization. The drop rate in the algorithm takes the form of Packets Per Drop (`PPD`) or `DPR`, and these two can be interchanged using Equation 4.14.

$$DPR = (D * B + Q)/PPD \tag{4.14}$$

Equation 4.14 is derived from the equations in Section 4.1.3. The equation is modified to account for the queue level Q that is present. A sustained Q means that $D * B + Q$ packets arrive at this link each round-trip. Each round-trip, the link can carry $D * B$ of those packets.

2RegionRED thus aims to keep the queue more or less steady by making these small incremental adjustments of `dQ/W`. `dQ` is computed simply from the difference in the size of the estimated queue at the beginning and end of an interval. The estimation of `W` comes from Equation 4.5. It uses the reasoning that if a particular drop rate is applied over a long period of time, the average value of `W` of all the competing TCPs is roughly:

$$W = \sqrt{(2/3) * PPD} \tag{4.15}$$

In the algorithm, however, the drop rate does not remain fixed, but increases and decreases in response to a growing or shrinking queue, respectively. In a steady state system, the drop rate will increase in successive intervals in reponse to a rising queue ($dQ > 0$). Eventually, it is almost always the case that the drop rate will overshoot that value which is necessary to steady the queue, and the queue will begin to drain ($dQ < 0$). Thus there is a space in which the drop rate hunts up and down around the ideal drop rate, and these oscillations are matched by oscillations in the queue. Smaller oscillations indicate that the algorithm is behaving with greater responsiveness and/or accuracy.

Because PPD is not fixed, Equation 4.15 should not be strictly applied. The estimation of `W` according to Equation 4.15 will cause its value to jump as frequently as `PPD` changes. `W` will increase and decrease as the drop rate hunts, over- and undershooting the ideal drop rate. For this reason, the algorithm presented includes a simple EWMA smoothing of W. In the steady state example, this will cause the value of W to remain closer to that which corresponds to the `PPD` associated with the middle of an oscillation.

As mentioned earlier, the High-N algorithm of 2RegionRED adjusts the drop rate according to an implicit estimate of the number of flows. This implicit estimate results from the relationship between PPD and the average W per TCP that results (Equation 4.15), and the relationship between W and N (Equation 4.4).

Tuning the algorithm There is a time delay between the moment a packet is dropped and the time the corresponding window reduction is reflected in the queue size. A scheme such as the one proposed that makes fine local adjustments to the drop rate according to successive "deltas" should be sure to take this into account. However, we run into two problems.

1. From the earlier explanation, note that it is only by the *end* of T2 that we can measure dQ and assess what the value of the new drop rate should be. Thus, the value of

newDPR is adjusted from that used during *T1*, but is applied to *T3*. However, what drop rate should be applied in T2?

2. The analysis used here assumes all connections have RTTs of 100ms. In reality, competing TCPs may exhibit a range of RTTs. Other delays from multiple bottleneck paths may also impact the time it takes for the consequence of a packet drop to be reflected at the queue. Additional complexity is introduced by flows arriving and leaving with variable frequency, as well as the exponential start-up dynamics of new TCPs and ACK-compression effects.

Because of the variability and number of unpredictable factors that can arise from (2), it is not worthwhile and perhaps impossible to predict what the "best" drop rate adjustment should be. Thus, an approximate should suffice. Because we would also like a simple and reasonable way of dealing with (1), we introduce an `effectivePPD` value that is the average of the Packets Per Drop applied within the past two intervals.

$$effectivePPD = 0.5 * PPD + 0.5 * PPD_{old} \tag{4.16}$$

We adjust the drop rate based upon `dQ` and this value of `effectivePPD`.

The drop rate is further constrained between 1/DB (roughly one drop per RTT) and 1/10. A drop rate of 10% (or perhaps even larger than 10%) should be sufficient to handle worst-case problem nets. One drop per RTT marks the boundary between the Low-N and High-N Regions. If a smaller drop rate is sufficient, then the queue should fall back into the Low-N Region's control. Thus, 1/DB is a sufficient lower bound on the drop rate in the High-N Region.

Pushing the queue to target The simple method just described ties a drop rate to some correlation between the number of drops and the *change* in the queue. This is very different from tying the drop rate to a particular queue size, as traditional RED does. Because 2RegionRED looks at a succession of dQ's, the resulting queue size could drift all over the place.

If the algorithm functioned properly, it could keep the queue size relatively steady, but it might keep this steady queue size at a level far elevated above Bflat.

It is for this reason and in the interest of avoiding excessive queuing delays, that 2RegionRED incorporates the idea of a *target value*. As mentioned earlier, we chose Bflat to be the target value. At every interval, the algorithm examines `distToTarget`, that is, how far away the current queue size is from the target value. Using the same rationale as earlier in this section, the number of extra drops it would take to push the queue to target is roughly `distToTarget / W`. We claim it is not necessary to push the queue back to the target immediately. It is sufficient and less prone to instability if these additional adjustments are applied gradually.

We make a simple addition to the algorithm by increasing the number of drops applied in the next interval by a fraction of this value, according to the following equation.

$$extraDrops = \left(\frac{q\_est - target}{Bmax - target}\right) * (distToTarget/W) \tag{4.17}$$

This increases the fraction of drops linearly with the size of `q_est`, the estimated queue size. Note that many other schemes or variations of the above will also accomplish the same task of gradually pushing the queue to the target value.

Simulations demonstrating the High-N Region can be found in Section 6.3.

4.3.3 Fundamental limits to stability

The analysis used in constructing the 2RegionRED algorithm assumes that the TCP flows are all operating in congestion avoidance. Real networks, however, involve much more complicated dynamics. Some commonly occurring phenomena that can cause the algorithm to behave less than ideal are described below. In Chapter 6, we observe the consequences through simulation when 2RegionRED is used in their presence.

Consequence of TCP slow start

As shown in Figure 2-1, when a flow begins, it enters a slow start phase where it probes for bandwidth before reaching an equilibrium. The exponential growth during the slow start

Every interval I:

```
distToTarget = q_est − target;
dQ = q_est − q_est_old;
effectivePPD = 0.5 ∗ PPD + 0.5 ∗ PPD_old;
W = 0.9 ∗ W + 0.1 ∗ (sqrt((2/3) ∗ effectivePPD));
//Compute extra drops to drive the queue to target
if (distToTarget > 0)
        extraDrops = (distToTarget/(Bmax − target)) ∗ (distToTarget/W);
else
        extraDrops = 0;
//Compute new drop rate
origDPR = (D ∗ B + q_est_old)/effectivePPD;
dDPR = dQ/W + extraDrops;
newDPR = origDPR + dDPR;
ppd0 = (D ∗ B + q_est_old)/newDPR;
//Constrain PPD within reasonable limits
if (ppd0 > D ∗ B)
        ppd0 = D ∗ B;
if (ppd0 < 10)
        ppd0 = 10;
//Update or reset variables
q_est_old = q_est;
PPD_old = PPD;
PPD = ppd0;
Pdrop = 1/PPD;
```

Figure 4-7: Pseudocode for High-N Region.

phase can easily fool 2RegionRED into thinking that there are far more flows than there actually are, causing the algorithm to impose higher drop rates than is necessary. This will cause not only the slow-starting flows to slow down, but will take down other flows as well. If the drop rate is excessive in comparison to the number of active flows, the queue will drain and the link will become underutilized. If the system experiences many timeouts, the poor performance triggered by slow starts may be perpetuated. [4]

Consequence of Two-Way Traffic

Zhang, Shenker, and Clark identified two phenomena that can arise when reverse traffic is added into a system [32]. ACK-compression (described in Section 2.3) is characterized by a highly fluctuating queue. These wild fluctuations break the association between the drop rate, number of flows, and queue size, which 2RegionRED relies on in order to adjust the drop rate effectively. A second phenomena termed "out-of-phase window-synchronization" results in degraded link utilization.

Consequence of Heavy Tailed Distributions

Recent studies have characterized various aspects of Internet traffic to be *heavy-tailed*. Connection sizes or durations have been suggested to be characterized by heavy tailed or log normal distributions [17]. For instance, the large majority of transfers, including web transfers, are small and represent a small percentage of all packets. Meanwhile most of the packets on the Internet belong to a very small percentage of flows, such as some very large FTP transfers. If transfers are large, dropping a packet will have the desired effect of relieving congestion (the TCP will halve its sending window). On the contrary, many small transfers finish their transfers while still in slow start phase. Dropping a packet from such a small transfer may cause the retransmission of the single dropped packet one RTT later, upon which the connection immediately terminates because all other packets have already arrived at the destination. Such a packet drop does not help in alleviating congestion and

[4] We do not explore mechanisms that distinguish slow starting flows from those that have already determined their appropriate sending rate (and have set `ssthresh`). Dealing with such flows differently may provide a simple solution to this problem of determining the proper drop rate and is worth investigation.

furthermore reduces the throughput of the small flow considerably.

Hard Limits

In order for 2RegionRED to work, or many other algorithms for that matter, there is an assumption that the frequency of change in the system environment is limited. If change occurs frequently within a sub-RTT, then we cannot make any sensible correlation between the dropping probability in a previous interval and the change in queue size in the current interval. The result of a drop rate applied at one instance becomes masked by transients, flows slow-starting or leaving, etc.

It seems that schemes that keep a count of active flows would be more immune to less tractable things like short-lived transients and slow starts. However, if flows were extremely short-lived and the frequency and/or degree of change high, then even counting flows is of no extra help.

4.3.4 Discussion and Summary

2RegionRED is based upon the new RED signal which emphasizes sustaining as small a queue necessary that still allows for maximum link utilization.

It uses knowledge of TCP window dynamics to determine the appropriate drop rate for any N. When modeling long TCP flows in congestion avoidance phase, it is already well-established that a drop (or loss) rate p administered over a long period of time will result in TCP windows of size $W \sim 1/sqrt(p)$ [22]. Furthermore, because $W \sim 1/N$, it is understood that the drop rate applied should be $p \sim N^2$. Earlier in this chapter, we reformulated these relations.

In this chapter we discovered that there is a logical breakdown into two regions of operation dependent on the drop rate required ($> 1DPR$ or $\leq 1DPR$), and that each region requires a different mechanism. We found when investigating the Slope-Controlled scheme that it was difficult to come up with a simple and reliable algorithm that could easily indicate which of the two regions the algorithm should be operating in at any time. Backing away from the goal of minimum buffer occupancy, we found a reasonable compromise in

2RegionRED. Though 2RegionRED does not use the queue as its measure of load, it makes a clean divide of the queue into two distinct parts, which enables the algorithm to discern easily whether the Low-N or High-N Region's algorithm is to be applied. This lends itself to a simple attractive implementation, which trades off some queuing delay for maximum link utilization.

Comparison to FPQ

Of all the algorithms mentioned in the Related Work (Section 3.3), 2RegionRED is most similar to FPQ. Both congestion control algorithms associate load with the number of flows N. Both adjust the discard rate based upon N using similar theory derived from TCP congestion avoidance behavior. Whereas FPQ uses a bit vector to estimate the number of flows from which it directly sets the drop rate, 2RegionRED (in the High-N Region) looks at the impact of the drop rate on changes in the queue size to determine N implicitly.

Not considering what happens when N grows excessively large, FPQ's dropping probability function was also designed in a way that approximates the new RED signal introduced in Section 4.1.1. The size of its target queue varies with N according to the relation $DB/(2N-1)$, keeping in mind that it assumes TCP Tahoe flows. This is essentially identical to our formulation of the optimal buffer size ($W = 2DB/(3N-1)$. When N grows, this target queue shrinks and a drop rate proportional to N^2 is applied in order to keep the queue around the target queue value.

Establishing the dropping probability only on an estimate of N can lead to unpredictable buffer occupancy because the $p \sim N^2$ relation involves assumptions that do not account for flow start-up behaviors, varying RTTs and packet sizes, among other things. For example, one drawback of both algorithms is that they both rely on a guess of the average RTT (a canonical RTT). For these reasons, some algorithms though they decouple the dropping probability from the queue size still maintain a link between the two to ensure that the applied drop rate is in the right ballpark. 2RegionRED, for example, applies extra drops in order to push the queue to the target. The mechanisms and principles of FPQ are applied to other queue-based algorithms such as RED in order to keep the queue within reasonable limits.

The key difference then is that in 2RegionRED, we have explored the approximations that can be made and the novel mechanisms that can be used to enforce them, even if N is not explicitly tracked by the congestion control algorithm.

Chapter 5

Queue Estimation: Measuring the Persistent Queue

Up until this point, there has been little mention of how the queue is measured, or estimated. We discover that this is a critical part of the congestion control algorithms we have discussed. In this chapter, we explore three different queue estimation schemes, and analyze how well each is able to capture the value of the persistent queue. These three mechanisms, to be described in further detail, are:

EWMA EWMA (Exponentially Weighted Moving Average)

EWMA' EWMA on rising queue, but following the draining queue

ABSMIN Absolute-min method

We find that the ABSMIN method is most well-suited to tracking the persistent queue.

5.1 Persistent Queue Defined

The purpose of buffers is to absorb bursts. A queue estimation algorithm should not reflect transient information such as sub-RTT interarrival variation. It should, however, track the persistent queue, which is defined in the draft on RED-light by Jacobson, Nichols, and Poduri as "the queue occupancy that does not dissipate over a reasonable time period" [20]. The draft makes a clear distinction between finding the persistent queue and finding the average queue. Whereas a persistent queue tracks the number of packets that does not get cleared from the queue within some time scale of interest, averaging the queue involves taking the arithmetic mean of all the bursts over that period.

In measuring the level of persistent queue, it makes sense to define the time scale of interest to be on the order of a round-trip time, because that is the time scale over which TCP operates. In a RTT, a TCP relying on its ACK-clocking mechanism, can send a new window of packets. Also in a RTT, a TCP is notified about and can respond to congestion. Floyd, who proposed the use of EWMA in RED, also makes mention that it is ideal behavior if the estimated queue begin to reflect congestion only when the the measure of congestion has lasted longer than the average round trip time [13]. [1]

How can we verify whether a queue measurement mechanism is truly tracking the persistent queue? One qualification is that *when a persistent queue develops, the link should be fully utilized over that period.* In other words, we should expect the pipe to be full.

We use this fact in the following simulations to verify whether a queue estimation algorithm is indeed tracking the persistent queue or not.

5.2 Experiments with small N

We first experiment with the following simulation setup. TCP Reno connections are used.

SIMULATION small N

Topology: Single bottleneck dumbbell topology as in Figure 5-1, 155 Mbps bottleneck

Scenario: Infinite FTP transfers, with RTTs varying from 60ms to 140ms.

Direction $bn0 \rightarrow bn1$

From 0-10 secs, 10 flows start (Traffic is one-way)

Direction $bn1 \rightarrow bn0$

From 40-50 secs, 10 flows start (Traffic is two-way)

Simulation length = 80s.

[1] To be more precise, we should define persistent queue as the queue that does not drain within a timescale on the order of a propagation delay.

Unless specified all plots are shown for $bn0 \rightarrow bn1$ direction.

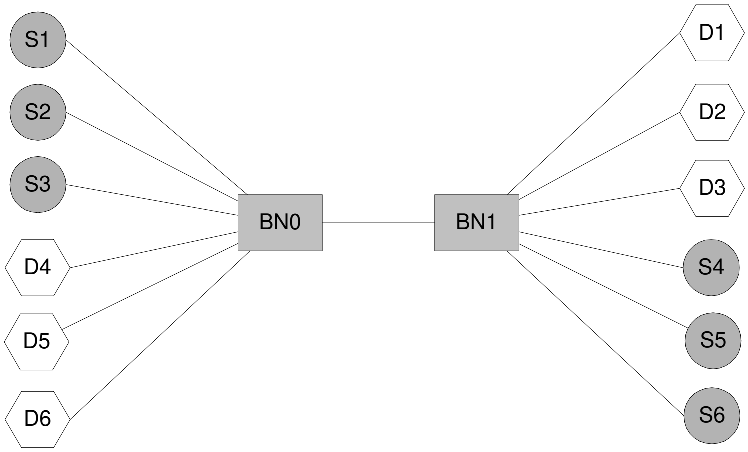


Figure 5-1: Single bottleneck topology. This bi-directional topology is used for all simulations in this paper.

We compare the following three scenarios:

- EWMA in RED
- EWMA' in 2RegionRED
- ABSMIN in 2RegionRED

5.2.1 EWMA

As described in Section 3.1, classic RED [16] uses the following low-pass filter to average the queue size.

$$aveq \longleftarrow (1 - w_q) * aveq + w_q * q \tag{5.1}$$

Figure 5-2 shows the behavior of EWMA in RED in a representative two second clipping in an early portion of the simulation where all traffic is one-way. Figure 5-3 shows the behavior

EWMA in RED

Figure 5-2: **One-way traffic**

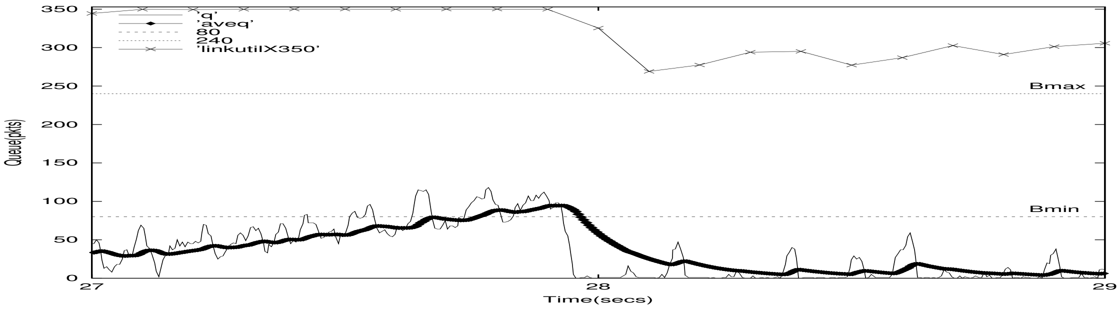
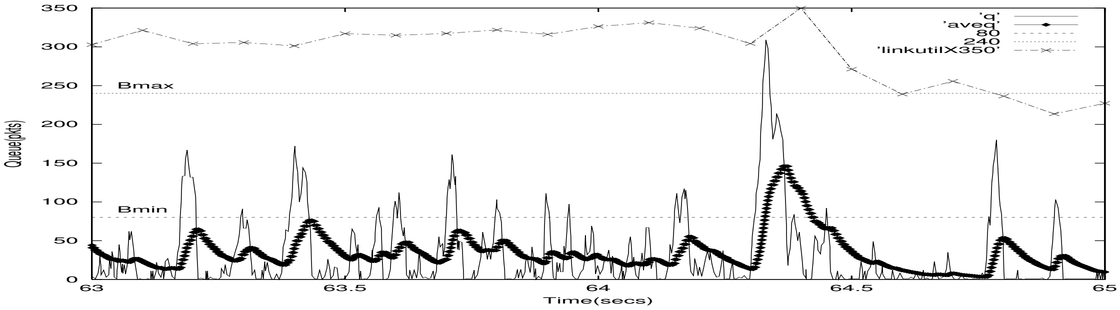


Figure 5-3: **Two-way traffic**



when reverse traffic is added. The plots display the average (`aveq`) and instantaneous (`q`) queue sizes, along with link utilization.

The RED parameters used in the simulation are: (Bmin, Bmax, Pmax) = (80, 240, 1/20), $w_q = 0.0007$ (equivalent to a time constant of 100ms).

In Figures 5-2 and 5-3, the link utilization is superimposed on the plot, scaled such that 350 represents 100% link utilization. Notice that even when the link is underutilized, the average queue indicates a positive value. If it were tracking the persistent queue, the estimator should have indicated an average queue value of zero. This is particularly obvious in the two-way traffic plot. It is easy to see that because EWMA is an averaging scheme, it does not track the persistent queue. EWMA is therefore not well-suited for 2RegionRED. It is used in RED, however, to smooth out bursts so that bias against bursty traffic is alleviated.

5.2.2 EWMA': Following the downward queue

We next make a small modification to EWMA, and introduce a second queue estimation scheme, EWMA'. This method was introduced in RED-light [20] . The EWMA' queue estimation scheme performs EWMA only on the growing queue. Because bursts that are cleared within a round-trip are to be ignored, EWMA' *follows* a draining queue downward. This is summarized as follows.

On a growing queue:

$$aveq \longleftarrow (1 - w_q) * aveq + w_q * q \tag{5.2}$$

On a draining queue:

$$\textit{If (aveq < q)} \quad aveq \longleftarrow q \tag{5.3}$$

Figure 5-4 and Figure 5-6 shows EWMA' in 2RegionRED operating in the Low-N Region for one-way and two-way traffic. Note that $N_{1DPR} = 30$ (See Table 8.1) while this simulation involves at most 10 flows in either direction).

For this 155Mbps bottleneck simulation, the 2RegionRED values used are: (Bmin, Bflat, Bmax) = (89, 134, 1292). See Table 6.1. A gain of $w_q = 0.0007$ (equivalent to a time constant of 100ms) is used for the EWMA' averaging.

Though we have shown EWMA as it operates in RED while demonstrating EWMA' in 2RegionRED, the contrast that results from EWMA' following the downward queue is obvious in the plots.

Also notice the effects when reverse traffic is added in Figure 5-6 ACK-compression creates a highly fluctuating queue length. The EWMA on the growing queue follows these bursts, which often burst above Bmin for a short time. At Bmin, a single packet is dropped each time in accordance with the 2RegionRED algorithm. It is not unlikely for the average queue to swing over then under Bmin with each of these bursts, creating far more drops than desirable, and leading to poor underutilization particularly when the number of flows is small.

Queue estimation schemes with Small N in 2RegionRED

Figure 5-4: **EWMA', One-way traffic**

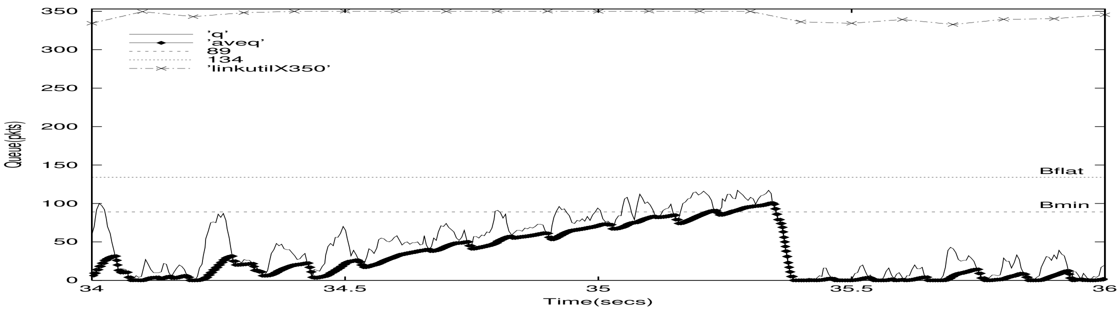


Figure 5-5: **ABSMIN, One-way traffic**

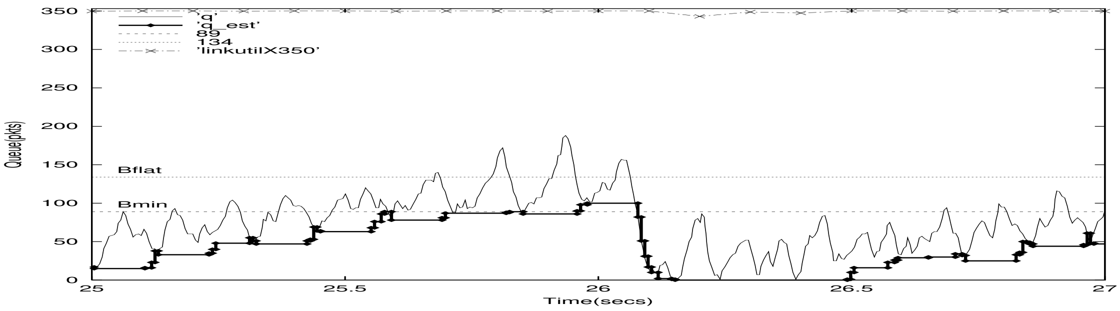


Figure 5-6: **EWMA', Two-way traffic**

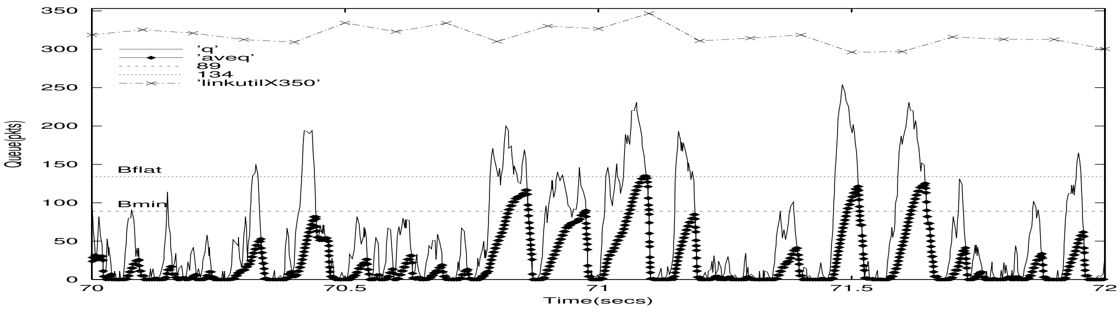
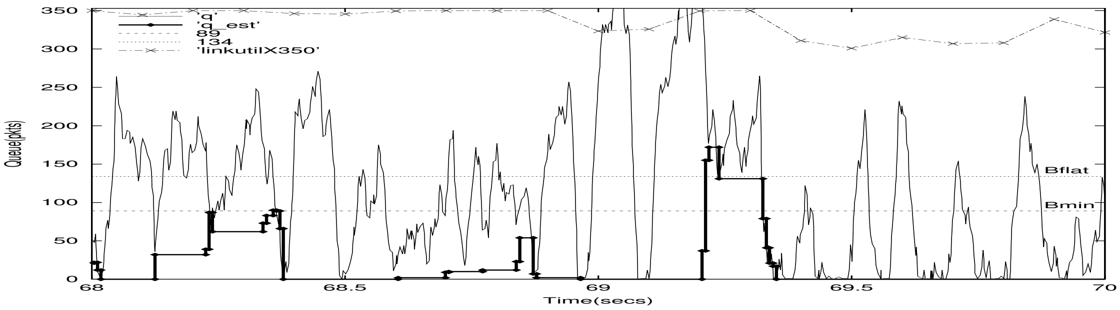


Figure 5-7: **ABSMIN, Two-way traffic**



Different estimation schemes: Entire Simulation

Figure 5-8: **EWMA in RED**

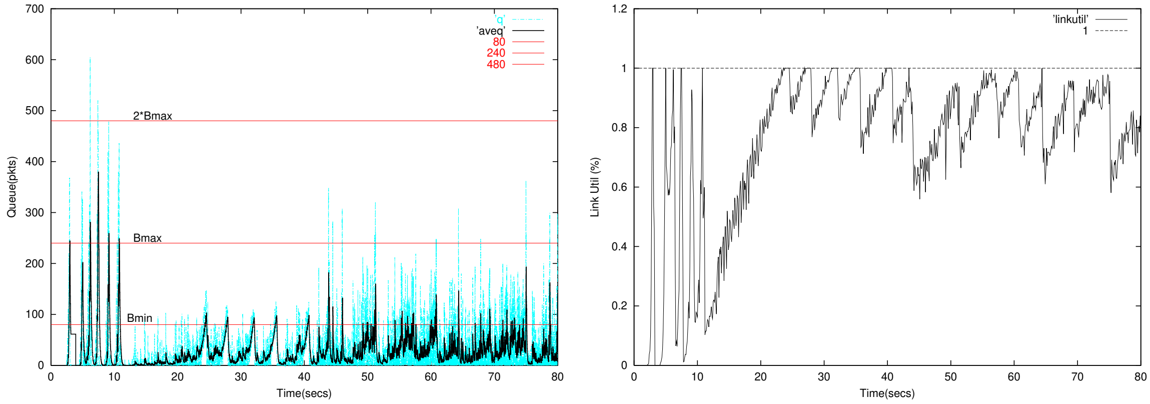


Figure 5-9: **EWMA' in 2RegionRED**

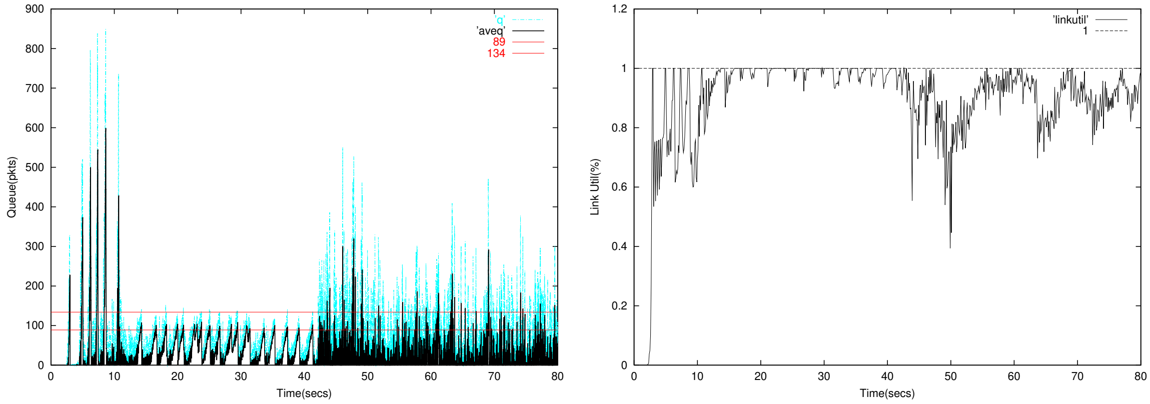
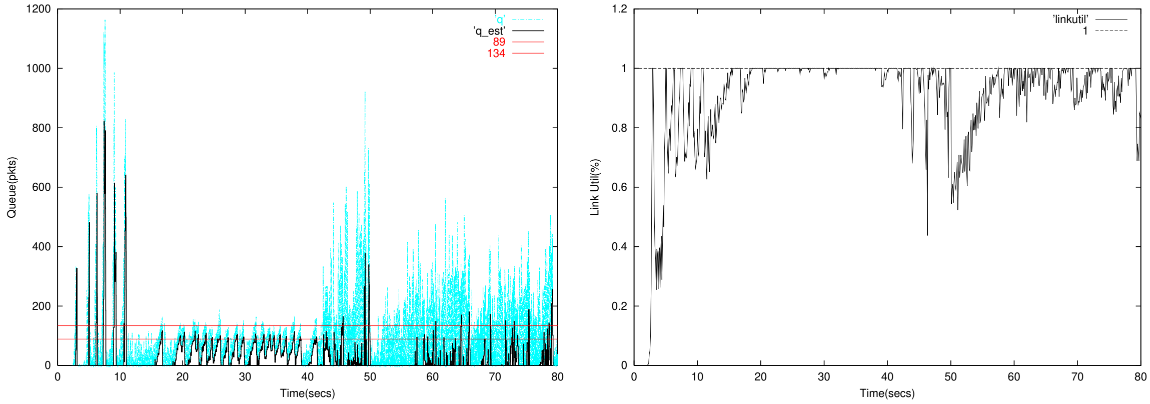


Figure 5-10: **ABSMIN in 2RegionRED**



We see that with one-way traffic, EWMA' works well in 2RegionRED. However, because of the highly fluctuating queue, EWMA' does not work well with the 2RegionRED algorithm in the presence of two-way traffic in these small N scenarios. [2]

5.2.3 ABSMIN: Taking the Absolute Min

If we take the strict definition of persistent queue, then "[a] queue that is empty at least once every round trip time should result in a queue size of zero." [20, Page 13] A straightforward method of measuring the persistent queue is to take *the absolute minimum value of the instantaneous queue over some time interval.* This time interval, referred to as the ABSMIN interval should be at least as large as the assumed average round-trip, such that most sub-round-trip variants are excluded from the measurement. Because the router does not know the average round-trip value, we assume a canonical value of 100ms.

In the implementation, the instantaneous queue size (`q`) is updated once per packet in the packet handling routine. The simplest implementation of ABSMIN would involve updating the queue estimate once every 100ms with the smallest instantaneous queue size seen within that period. However, much can happen within 100ms. In order for the algorithm to be more responsive to changes in the queue, the implementation used in this report updates the estimated persistent queue (`q_est`) more frequently, fifteen times every 100ms (or once every 6-7ms) In order to retain knowledge of the past 100ms, the implementation also keeps a small array of size fifteen, which stores the smallest value of the instantaneous queue seen within each of the fifteen most recent 6-7ms intervals. Each interval, the queue estimator scans the array for the smallest value and this is assigned to the value of the current persistent queue. Each interval, the oldest array element is also replaced with the most recent information.

The results for both the one-way and two-way cases are shown in Figures 5-5 and 5-7 respectively. The same parameters for 2RegionRED that were used in the EWMA' simulation are also used here.

[2] For the same reason (the use of a deterministic drop at Bmin), we conjecture that EWMA' does not work well in the RED-light algorithm [20] (for which it was originally proposed) when there is bi-directional traffic and a small number of flows.

5.2.4 Consequences of Lag in ABSMIN

By tracking the absolute minimum queue, we allow the queue to do its job in absorbing transient bursts. The main drawback in using ABSMIN is that because we look over the *past* 100ms to find a new minimum, there is sometimes a lag in identifying the persistent queue. This lag can be easily perceived within the plots, most noticeably when the queue is rising. Only when ABSMIN recognizes that such a high queue has persisted for at least 100ms does it perceive the rising queue to be persistent, rather than a transient. The consequences of the lag are twofold:

1. The instantaneous queue is allowed to grow high and unchecked over intervals smaller than the ABSMIN interval, even if it is *not* a transient. This can be undesirable if within this time the queue length exceeds the buffer capacity, causing massive drops.

2. The 2RegionRED algorithm may not respond as quickly or accurately in adjusting the drop rate because it is acting on information that is roughly a round-trip time old.

Finding a simple improvement on ABSMIN to avoid negative consequences of the lag is a topic for further research.

5.3 EWMA' vs. ABSMIN

An interesting observation about ABSMIN is that though there is some lag in tracking the persistent queue when the queue is rising, the estimated queue follows the draining queue almost immediately when its value drops below the previously estimated queue size. This behavior of following the downward queue was seen previously in EWMA', for it is the defining characteristic of in EWMA'. Observing ABSMIN's similar behavior as a result of tracking the absolute minimum over an interval provides some rationale for why the mechanism of following the downward queue in EWMA' was originally added to better track the persistent queue.

When considering EWMA' and ABSMIN in 2RegionRED, the most visible difference occurs in the two-way traffic plots. Observe, in Figures 5-9 and 5-10, the behavior of the instantaneous (in gray) and average queue sizes (in black) in the second half of the simulation after reverse traffic has been added. EWMA' indicates the frequent existence of a persistent queue. In the ABSMIN plot, the persistent queue makes a much rarer appearance.

At first sight, it seems that this difference may have simply resulted from setting the gain w_q in EWMA' too high, or from assuming in ABSMIN a canonical round-trip that is too large. (Recall that the canonical round-trip of 100ms is used to determine the interval, also 100ms, over which the persistent queue is measured.) Suppose the gain in EWMA' were smaller, or the ABSMIN's interval too short. Is there any resulting difference between these two mechanisms?

For EWMA', even if the gain appears to cause the estimator to follow transient bursts too closely, decreasing the gain such that the average tracks the instantaneous queue more slowly is not a good solution. A gain that is too small will cause the estimated queue to lag too far behind in situations where the persistent queue is indeed rapidly growing.

As for ABSMIN, in the next section we observe what happens when the canonical RTT is much less than the actual average RTT.

5.4 ABSMIN: Different RTTs

Large average RTT In this section, we run the same simulation but change the average RTT of the flows to roughly 200ms. Flows may have a round-trip of anywhere between 170ms and 230ms. The ABSMIN algorithm is unchanged, however, and uses what now becomes a poor setting of the interval by looking at the past 100ms of history to determine a minimum. Figures 5-13 and 5-16 shows the result. ABSMIN begins to bear some resemblance to EWMA' in that it follows sub-RTT transients. Whereas EWMA' uses an averaging that causes the queue to gradually overshoot the persistent value, ABSMIN is characterized by a queue that rises 1) more steeply following the general slope of the burst, and 2) after an initial lag period.

In two-way simulations, ABSMIN proves superior to EWMA', even if the actual RTT is 200ms and ABSMIN uses an interval of 100ms. But this is largely a consequence of reverse-traffic dynamics causing bursts to be smaller and very short-lived. As a result, most transients are not reflected in ABSMIN's estimation, as desired.

ABSMIN in 2RegionRED: Different RTTs, One-way traffic

Figure 5-11: **Average 41ms** [25ms-58ms]

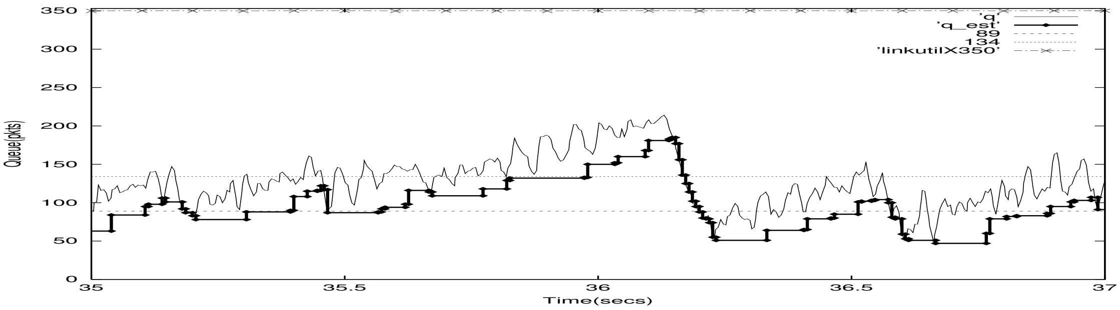


Figure 5-12: **Average 103ms** [71ms-136ms]

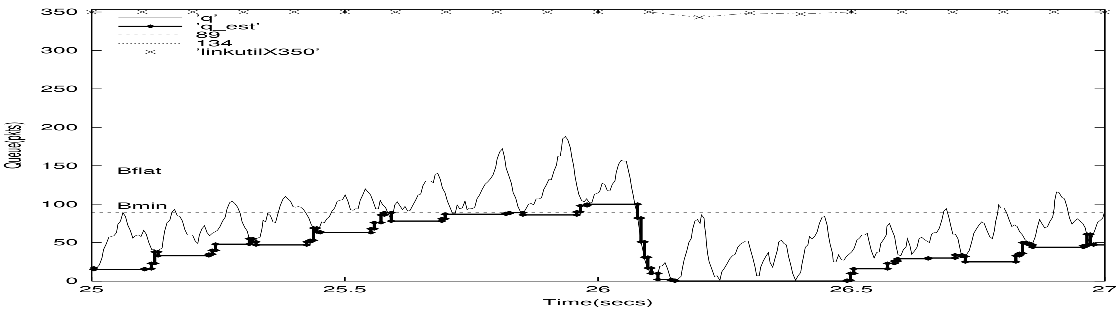
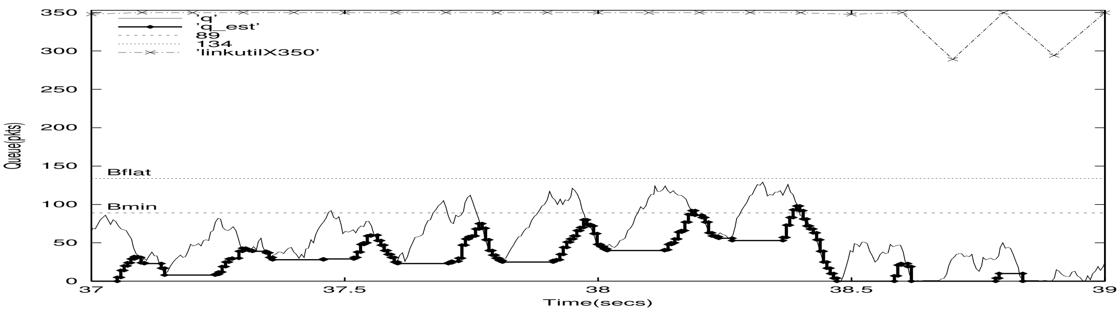


Figure 5-13: **Average 204ms** [178ms-227ms]



ABSMIN in 2RegionRED: Different RTTs, Two-way traffic

Figure 5-14: **Average 41ms** [25ms-58ms]

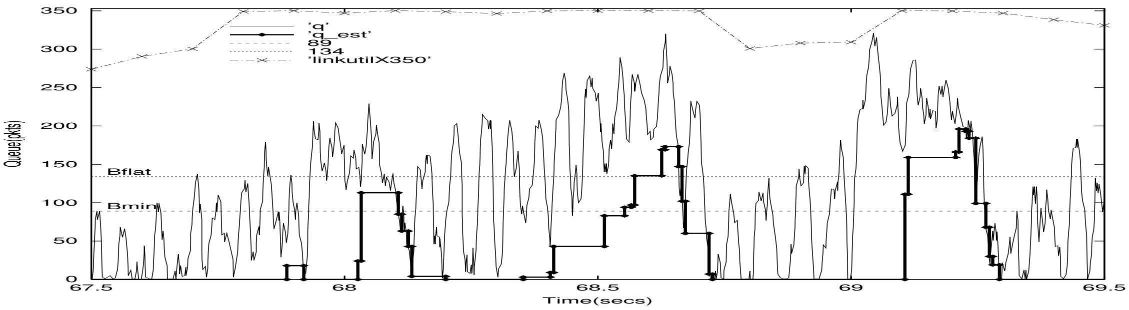


Figure 5-15: **Average 103ms** [71ms-136ms]

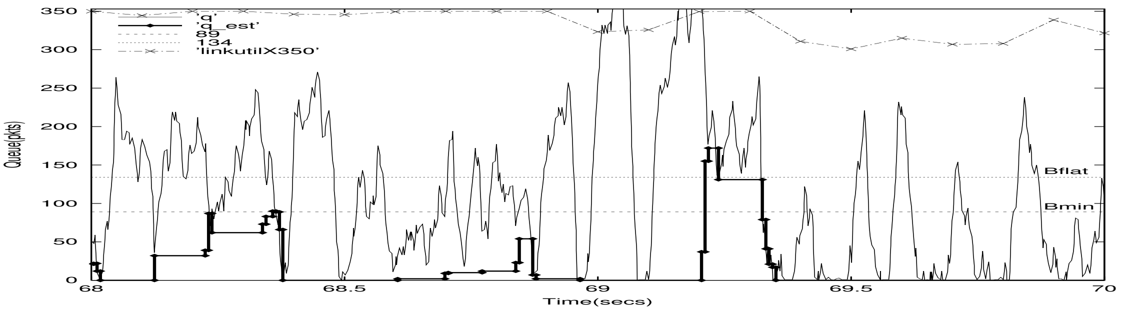
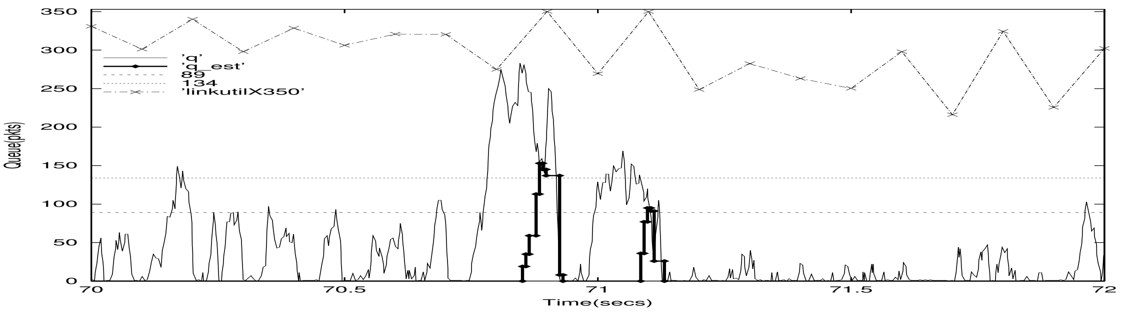


Figure 5-16: **Average 204ms** [178ms-227ms]



Small average RTT Figures 5-11 and 5-14 show what happens when the average RTT is much smaller than 100ms. The same simulation is run, but with an average connection round trip of 41ms. Because of the well-known RTT bias towards flows with shorter RTTs, the queue rises more quickly per unit time. As a result, the deterministic drop at Bmin is often unable to push the queue back into the Low-N region where it "belongs". From the plots, we see that this excursion is quickly corrected by the drop rate applied in the High-N Region.

5.4.1 A note on the "average RTT" of the system

In several of the simulations, we specify what the *average RTT* of the flows across a bottleneck is. This value is an approximation. It reflects the propagation delays of the flows, and does not incorporate any queuing delays that arise at congested routers. It also reflects the *general* properties of the system, because different flows are active at any given time.

By *average* we mean that if N flows were increasing their flows in congestion avoidance, the rate of increase is equivalent to N flows that all share the same (the "average") RTT.

For example, suppose there are two flows and flow 1 has a RTT of 100ms, flow 2 has a RTT of 50ms. By average RTT we do *NOT* mean the strict mean, (50ms+100ms)/2 = 75ms. In one second, flow 1's cwnd can grow by 10 packets, while flow 2's cwnd can grow by 20 packets, which is a total of 30 packets per second. This is equivalent to two flows that each increase their windows by 15 packets per second. A flow that increments its cwnd 15 times per second has a RTT of 1000ms/15 = 67ms. Thus, we say that the average RTT in the system of the 50ms and 100ms flow, is 67ms, and not 75ms.

5.5 Experiments with large N

The plots shown thus far reveal the differences among different queue estimation schemes operating with a small number of flows. For a 155Mbps bottleneck, and RTT of 100ms, the $N_{1DPR} = 30$ flows. In this section we show the EWMA' and ABSMIN mechanisms as they operate in 2RegionRED's High-N Region, according to the following scenario.

Queue estimation schemes with Large N in 2RegionRED: One-way traffic

Figure 5-17: **EWMA'**

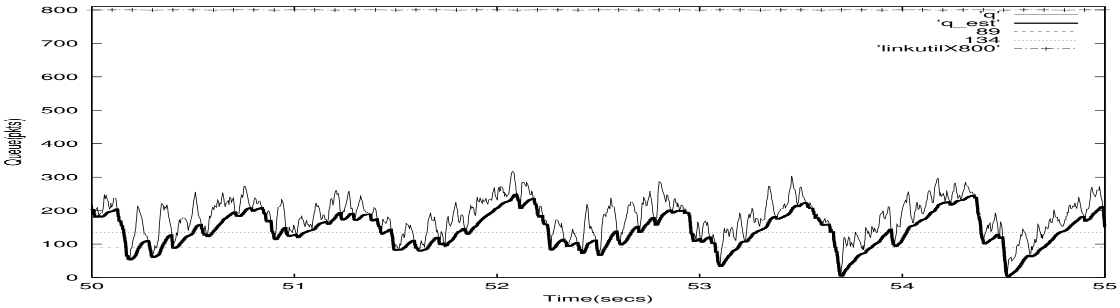
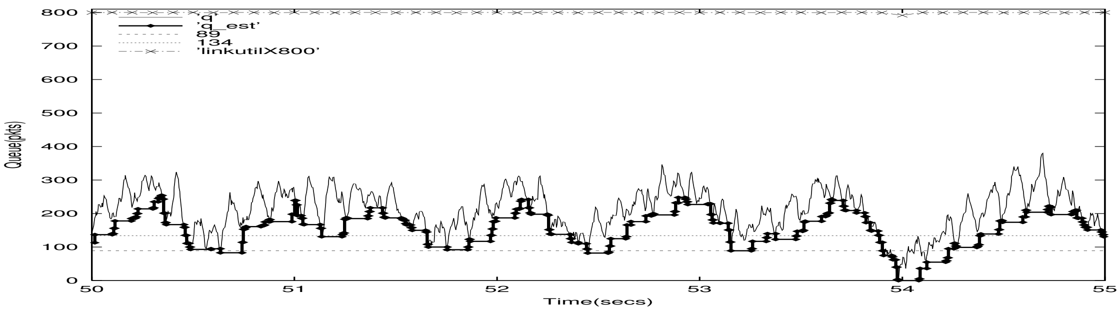


Figure 5-18: **ABSMIN**



SIMULATION large N

Topology: Single bottleneck, 155 Mbps

Scenario: Infinite FTP transfers, with RTTs varying from 60ms to 140ms.

Direction $bn0 \rightarrow bn1$

From 0-10 secs, 50 flows start (Traffic is one-way)

Direction $bn1 \rightarrow bn0$

From 100-110 secs, 25 flows start (Traffic is two-way)

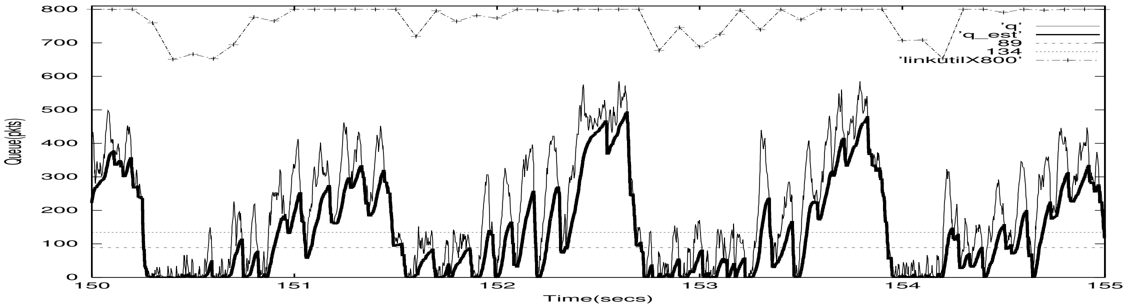
Simulation length = 200s.

The target value is Bflat, or 134 packets. Bmax is set to one delay-bandwidth product and does not affect this particular simulation.

The results can be found in Figures 5-17 through 5-20. For the two-way portion of the simulation, the queues in both directions on the bottleneck link are shown.

Queue estimation schemes with Large N in 2RegionRED: Two-way traffic

Figure 5-19: **EWMA', both forward and reverse bottlenecks**



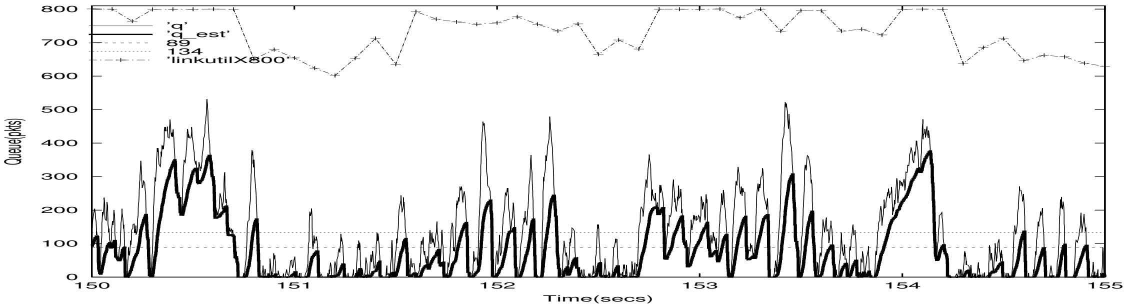
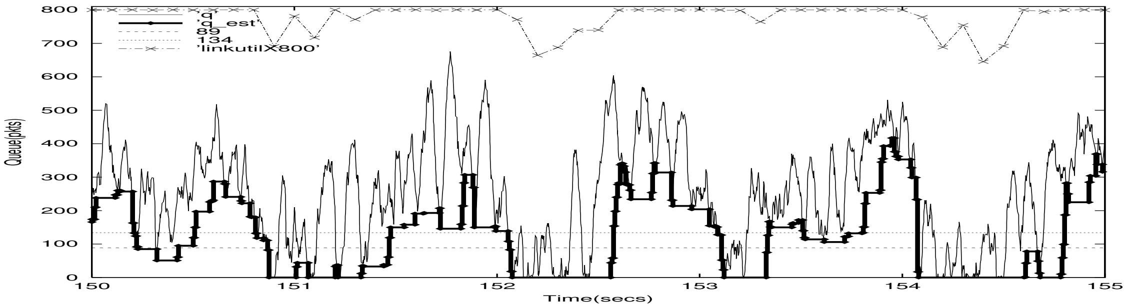
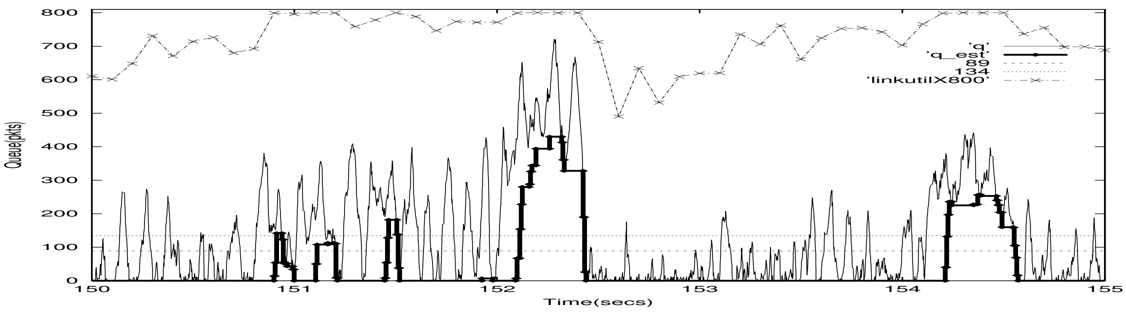


Figure 5-20: **ABSMIN, both forward and reverse bottlenecks**





In the one-way plots, there is no significant difference between EWMA' and ABSMIN. The bi-directional plots are more interesting. Notice again how ABSMIN does a better job of ignoring transient bursts than EWMA'. However, the link utilization of EWMA' is not any worse than that of ABSMIN for two reasons:

- Increased numbers of flows cause the queue to fill up more quickly.
- Because increased numbers of flows require a higher drop rate, the "accidental" drops at Bmin (when occasionally high bursts are absorbed into the queue estimate) have a smaller impact than they would in the small N scenario where the required loss rate is much lower.

Another observation is that for larger N, the signs of ACK-compression begin to appear in the two-way plots, characterized by the alternating queue pattern. We find that as the load or number of flows increases even further for this scenario, the existence of ACK-compression becomes very well-defined and resembles Figure 2-2. If such phenomena dominate the behavior in the queue, the difference between using EWMA' and ABSMIN (or EWMA for that matter) becomes even more blurred. In the next chapter we elaborate on the challenge this poses for 2RegionRED.

5.6 Summary and Discussion

Overall, we find that:

1. EWMA does not (and is not meant to) track the persistent queue. This is elaborated below.
2. EWMA' in spite of overshooting in following bursts, does a good job in tracking the persistent queue in one-way traffic, but is not well-suited to handle the bursty dynamics of two-way traffic. For a scheme such as 2RegionRED, where a packet is dropped deterministically at Bmin, transients that are taken to be persistent can cause low link utilization when the number of competing flows is small. When the number of competing flows is large, the extra drops at Bmin do not have a significant impact.

3. ABSMIN does a good job in tracking the persistent queue. However, it experiences a lag in tracking the persistent queue when the queue is increasing, and an inaccurate setting of the interval used in the ABSMIN queue estimation may cause some instances of less than ideal performance. When the interval is set too small, ABSMIN may exhibit similar behavior as EWMA' in following transients. When the interval is set too large, some persistent queue is not captured, and the consequences of the lag outlined earlier can be aggravated. However, the simulations have revealed that ABSMIN exhibits similar if not better behavior as EWMA' in one-way traffic while proving to be far superior to EWMA' in ignoring the frequent transients that appear when reverse traffic is present.

4. With larger N and reverse traffic, ACK-Compression may begin to mask the differences between the 2RegionRED performance that results when EWMA' or ABSMIN is used.

From these conclusions, we adopt ABSMIN as the method that best tracks the persistent queue. This queue estimation scheme is used in the following chapter to test the performance of 2RegionRED.

It is worthwhile to note that EWMA is used in RED because of an arguable trade-off between high throughput and lower average queuing delay [9]. Because of the bursty nature of traffic, one might imagine the following scenario. The scenario begins with an empty queue. Subsequently, N packets arrive back-to-back to this queue. A transient queue is formed and gradually drains, leaving the queue idle once more. This pattern is then repeated again and again, resulting in a highly oscillating instantaneous queue. Floyd believes that such a highly oscillating queue should be detected as congestion, but only because it perceives the high average queuing delays to be undesirable [9].

However as mentioned earlier, this thesis upholds the view that all such transients should not be detected as congestion and should be absorbed by the queue. In ABSMIN, these oscillations, if happening on a timescale less than a RTT, are not detected as congestion as desired.

Chapter 6

2RegionRED, Performance

In the previous chapter, we have already caught a glimpse of the behavior of the 2RegionRED algorithm. In this chapter, we present more extensive simulation results of 2RegionRED in some more realistic scenarios using the ABSMIN queue estimation scheme.

For comparison and contrast we also simulate two other adaptive algorithms, FPQ [27], and the version of Adaptive RED proposed by Floyd, Gummadi, and Shenker [15]. All simulations use the NS simulator [25], version ns2.1b8a. Code and detailed parameter settings used for the simulations can be found at http://ana.lcs.mit.edu/[XXX website]. Goodput and link utilization are used to evaluate performance.

We consider:

- One-way and Two-way traffic
- High-speed bottleneck bandwidths (up to 1Gbps)
- Distribution of transfer sizes

The simulations presented consider single bottleneck scenarios only and involve two bottleneck bandwidths, 155Mbps and 1Gbps. The 2RegionRED parameters that correspond to an N_{Bmin} of 10 is listed in Table 6.1 for each of these bandwidths. Recall that N_{Bmin} is used in 2RegionRED to set Bmin, and should enforce a drop rate of less than one drop per round-trip for a small number ($< N_{1DPR}$) of flows. Parameter settings for a 622Mbps bottleneck are provided for comparison.

BNBW	N_{Bmin}	Bmin	N_{1DPR}	Bflat	Bmax	D(secs)	DBP(pkts)
155Mbps	10	89	29	134	Varies	0.1	1292
622Mbps	10	357	58	446	Varies	0.1	5183
1Gbps	10	575	74	687	Varies	0.1	8333

Table 6.1: Parameters used in 2RegionRED Simulations. The tables that were used to compute Bmin can be found in the Appendix.

Parameter	Value
Packet Size	1500 bytes
TCP tick	0.001 secs
TCP version	Reno (unless otherwise specified)
ECN capable?	yes
gentle?	yes
wait?	yes
maxwin	infinite

Table 6.2: Parameters used in all Simulations (where applicable). The names of many of these variable names are specific to those used in NS [25]. *TCP tick* is the TCP clock granularity. All endhosts are ECN-enabled and the routers also use ECN. *gentle* refers to the gradual linear increase in drop rate between Bmax and 2*Bmax (See Section 3.1.2 and 4.3). *wait* indicates whether the router should wait between dropping packets. Uniform interdrop periods are used [16, Pages 10-11]. *maxwin* refers the the receiver's maximum window size for flow control and is set large enough so that it does not limit the sender's sending rate.

Congestion Control Algorithm	Queue Estimation Mechanism
2RegionRED	ABSMIN
Adaptive RED	EWMA
FPQ-RED	EWMA

Table 6.3: Queue Estimation Mechanisms used in the Simulations

6.1 Parameter Setting of Adaptive RED and FPQ-RED

The way in which 2RegionRED parameters are to be set is explained in Chapter 4. This section explains the parameter settings of two other adaptive algorithms: Adaptive RED [15], and Flow-Proportional Queuing (FPQ) [26], which are used for comparison to 2RegionRED. For FPQ, we use the FPQ-RED variation of the algorithm, as described in Section 3.3.3.

Adaptive RED Parameters Adaptive RED parameters are set as follows, according to suggested values [15]:

- **Bmin** = max(5, delayTarget * C / 2) which gives a target average queuing delay of delayTarget seconds. A delayTarget of 0.005 sec (5ms) is used here. C is the link capacity in packets per second.
- **Bmax** = 3 * Bmin
- **Pmax** is initialized to 0.05 or 5%.
- **gain** = 1 - exp(-1/C) Assuming a typical RTT of 100ms, this value of the gain is equivalent to a time constant of ten RTTs. See Section 3.1. Note that this is much smaller than the time constant used in RED and 2RegionRED.

FPQ-RED Parameters For FPQ-RED, Bmax and Pmax vary dynamically according to the targetLoss and targetQueue functions described in Section 3.3.3. Bit-vector flow counting is the mechanism which FPQ uses to estimate the number of active flows. Relevant parameters include v_{max}, the size of the bit vector, and t_{clear}, the interval over which bits in the vector are incrementally cleared. With the exception of the gain setting, Morris makes suggestions on how the other FPQ-RED parameters should be set [26]:

- **Bmin** = 5 packets
- **Bmax** = $targetLoss(q_t, N)$
- **Pmax** = $2 * targetQueue(N)$

- **gain** = 1 - exp(-1/C). C is the link capacity in packets per second. We have also experimented with smaller gains.

- **Bit-vector flow counting**: The parameters used are t_{clear} = 4.0 sec, v_{max} = 5000. (See [26, Pages 42–44])

6.2 Demonstrating the Low-N Region

We first demonstrate how the Low-N Region of 2RegionRED is able to control a system with few competing flows. We have already seen a glimpse of Low-N Region behavior in Chapter 5 with 155Mbps bottlenecks. In this section we show similar plots for a 1Gbps bottleneck. For a 1Gbps bottleneck and 100ms average RTT, the boundary of control for the Low-N Region is $N_{1DPR} = 74$ flows.

SIMULATION 1

Topology: Single bottleneck, 1Gbps

Scenario: One-way traffic, Long-running TCPs, RTTs (60ms-140ms),

000-003s: +10 flows ($N_{tot} = 10$)

100-105s: +30 flows ($N_{tot} = 40$)

150-155s: +30 flows ($N_{tot} = 70$)

For up to 70 flows (roughly the ideal limit of the Low-N Region).

Simulation length = 200s.

Figure 6-1 shows the steady state result when there are 10, 40, and 70 flows. The parameters used can be found in Table 6.4. When there are few flows, the single deterministic drop at Bmin is able to control the queue size without affecting link utilization. Notice the near 100% link utilization. These plots also reveal that as the number of flows increases, the more rapidly the queue will rise (marked by a steeper slope) as each TCP increases its sending window by one each RTT. As a result, the drop frequency also increases. Given any delay,

SIM 1: 2RegionRED

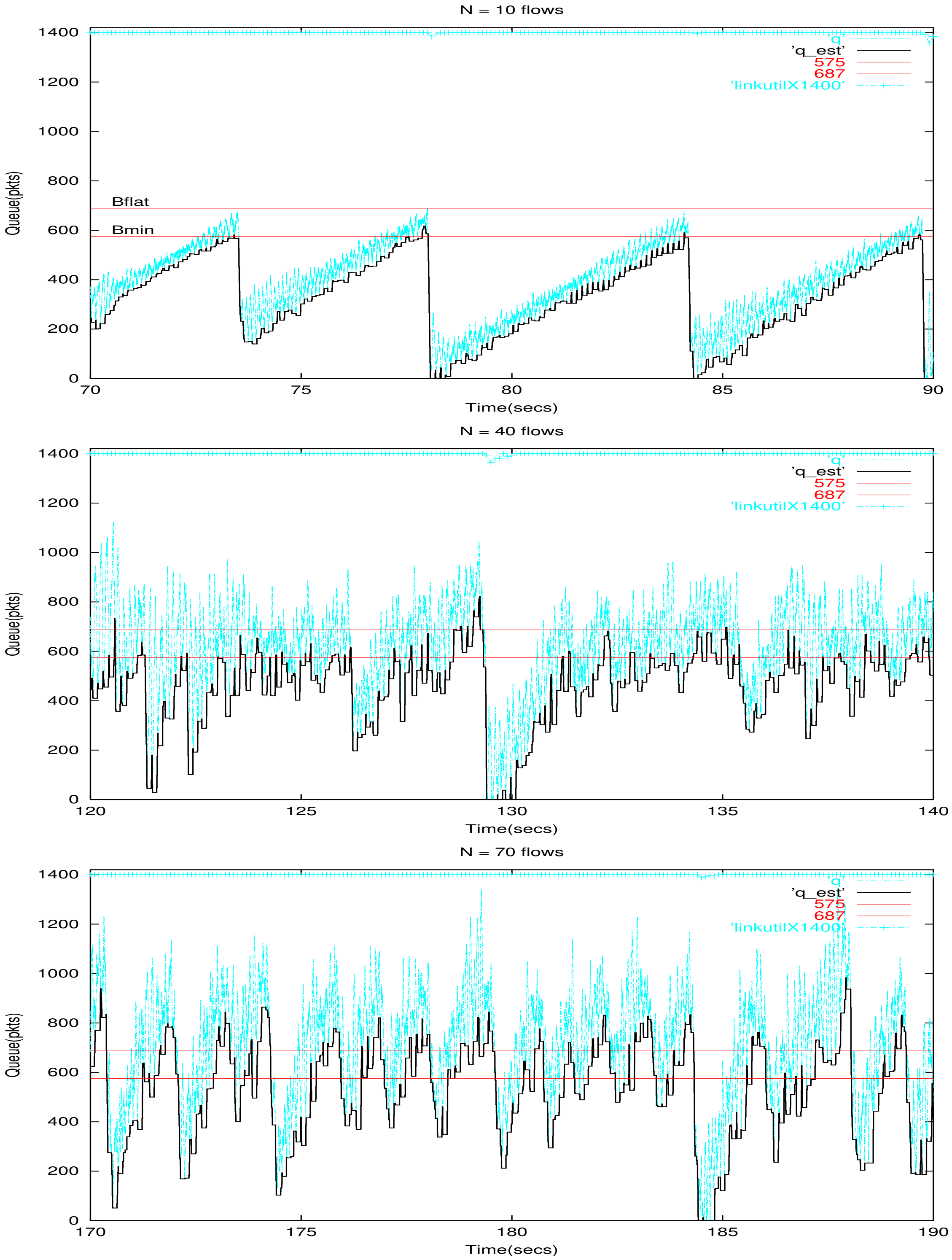


Figure 6-1: Queue behavior in the Low-N Region, $N_{1DPR} = 74$ flows.

bandwidth, N, and setting of Bmin, there is an implicit drop rate in the Low-N Region that can be calculated.

6.3 Demonstrating the High-N Region

For the same setup as above, we show the behavior of the queue in 2RegionRED, when up to 1000 flows are incrementally added.

SIMULATION 2

Topology: Single bottleneck, 1Gbps, 1.05Gb for all other links

Scenario: One-way traffic, Long-running TCPs, RTTs (60ms-140ms)

average RTT is 94.7ms for 1000 flows

000-010s: +100 flows ($N_{tot} = 100$)

060-080s: +200 flows ($N_{tot} = 300$)

130-150s: +200 flows ($N_{tot} = 500$)

200-220s: +300 flows ($N_{tot} = 800$)

270-290s: +200 flows ($N_{tot} = 1000$)

Simulation length = 350s.

The following pages demonstrate the behavior of the 2RegionRED, Adaptive RED and FPQ-RED congestion control schemes in this particular simulation.

2RegionRED in Figure 6-2 demonstrates very good behavior. Disregarding for now the noisy areas of the plot where new flows are introduced into the system, the adaptive drop rate keeps the queue near the target regardless of the number of flows (from 100 to 1000 flows). As the number of flows increases, the average window size of the competing TCPs should decrease and as expected the drop rate imposed by the algorithm should grow (proportionally to N^2). See Figure 6-3. W_{est} refers to the average reduction in window size of any TCP upon detecting a drop or congestion notification.

6.3.1 SIM 2: 2RegionRED

Parameter	Value
Bmin	575 pkts ($N_{Bmin} = 10$)
Bflat	687 pkts
Bmax	8333 pkts (DBP)
Btarget	787 pkts (Bflat + 100)
Blimit	8333 pkts (DBP)
guessed RTT	100ms
PPD initial	8333 pkts (DBP)
ABSMIN interval	100ms
Adjust Pdrop interval	96ms-144ms

Table 6.4: 2RegionRED Parameters for High-N Simulation. The *Adjust Pdrop interval* indicates how frequently the High-N algorithm in Figure 4-7 is called, which adjusts the drop rate. *PPD initial* is the initial setting of the drop rate, manifested in Packets Per Drop, when the estimated queue q_est first crosses Bflat. The ABSMIN interval uses a subsampling granularity of 6.7ms (See Section 5.2.3).

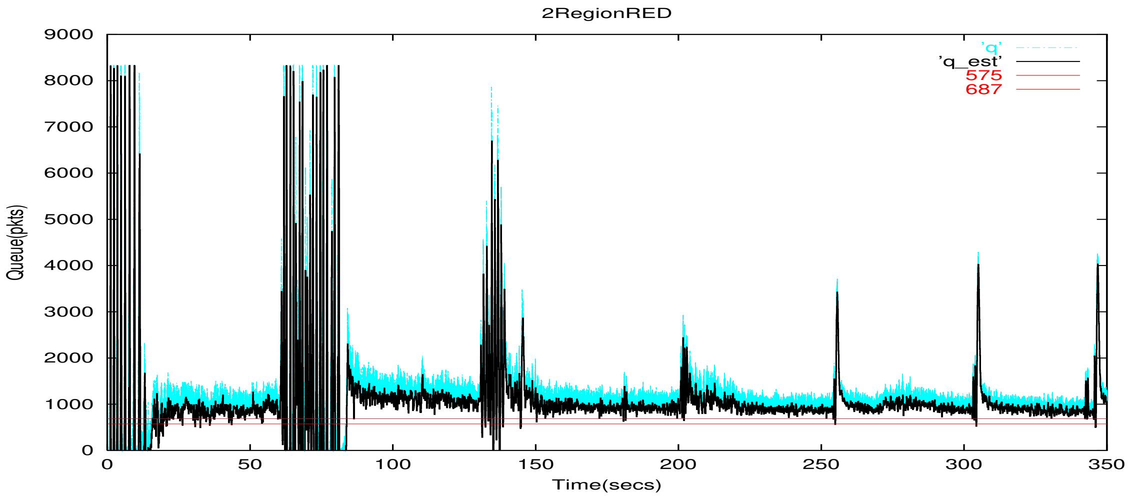


Figure 6-2: 2RegionRED Queue behavior in the High-N Region

SIM 2: 2RegionRED (cont'd)

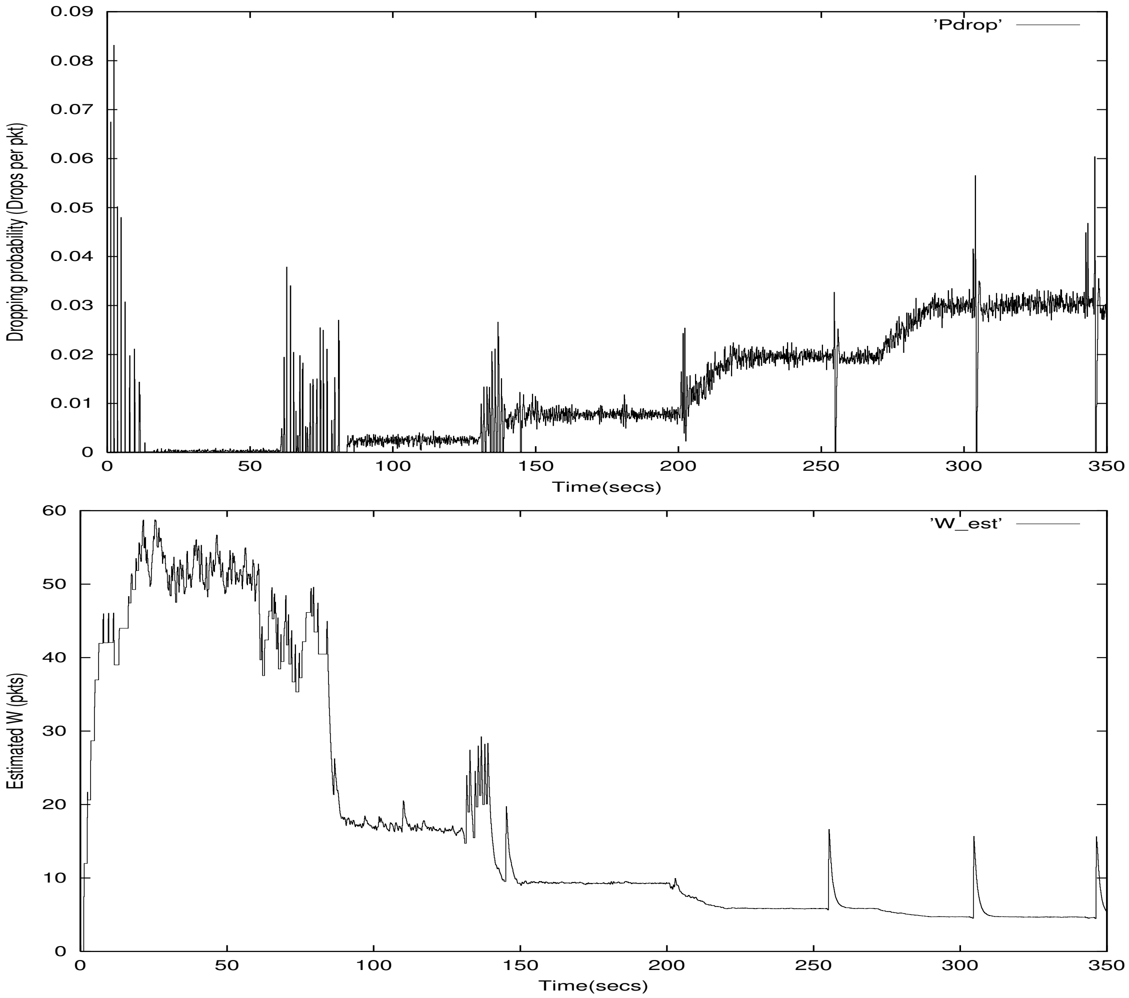


Figure 6-3: 2RegionRED plots: Pdrop and estimated W

SIM 2: 2RegionRED (cont'd)

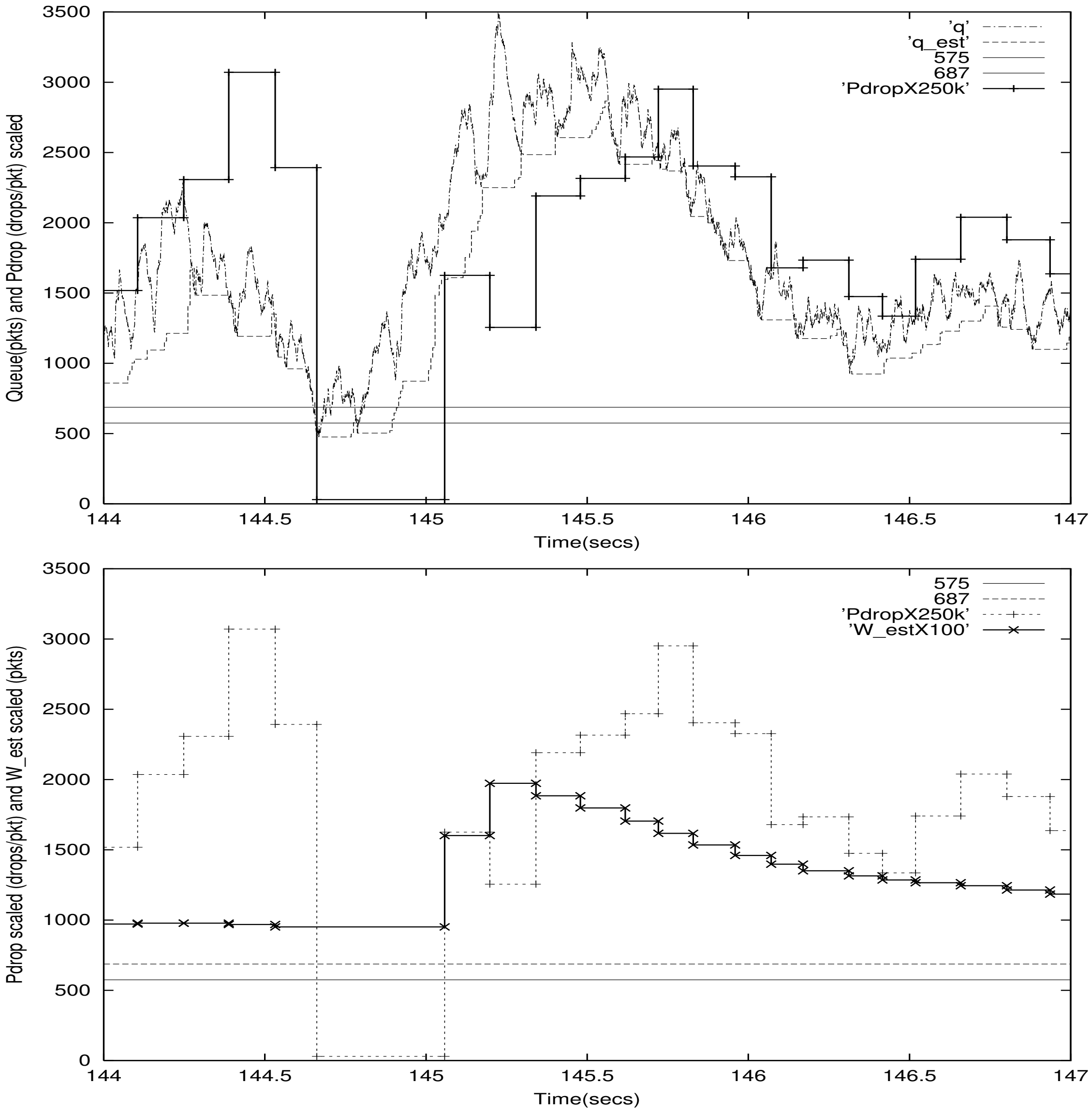


Figure 6-4: 2RegionRED: Crossing between Regions

The "Spike": Difficulties with Lag and with PPD_{init}

Notice the small spikes in the queue before the 2RegionRED algorithm brings it back under control. Figure 6-4 shows one of these "spikes" in more detail. This behavior can be attributed to the following two factors:

1. The consequence of the lag in the ABSMIN queue estimation method (See Section 5.2.4).

`q_est` lags the instantaneous queue `q` by roughly 100ms (the ABSMIN interval). Thus, when the 2RegionRED High-N algorithm on Page 67 measures `dQ`, it is observing slightly aged and thus less relevant information. The dropping probability will still adjust, though not as quickly as it might have given up-to-date information.

2. A PPD_{init} (PPD initial) that is too low relative to N. There is another factor that affects the time it takes for the dropping probability to stabilize the queue. When crossing Bflat from the Low-N Region into the High-N Region of operation, the drop rate is initialized to 1/P where P is the delay-bandwidth product ($PPD_{init} = P$). This drop rate is applied during one entire `Adjust Pdrop interval` before it is corrected. An initial drop rate that is too low will allow the queue to rise unchecked (possibly very high) during that interval. The sudden dips in the Pdrop plot in Figure 6-3 show where the drop rate is being reset to $1/PPD_{init}$. Recall that below Bmin there is no probabilistic dropping. This drop rate is not applied until `q_est` again crosses the Bflat threshold, which occurs a little above the 144.9 second mark in Figure 6-4. The low initial drop rates explain the spikes in the queue size. For larger N, the spikes are taller as one would expect.

It might seem that there is a simple fix in which the drop rate could be remembered for a short while each time the Bflat threshold is crossed from above. However this assumes that the traffic dynamics do not change on timescales less than or on the order of a few RTTs, an assumption of little certainty. Furthermore this one-way simulation is not a realistic representation of Internet traffic. We anticipate that such tweaks are not worthwhile when considering more complex and variable dynamics such as those present in Simulation 3.

6.3.2 SIM 2: Adaptive RED

Parameter	Value
Bmin	208.33 pkts
Bmax	625 pkts
2*Bmax	1250 pkts
Pmax	Variable
Blimit	8333 pkts (DBP)
gain	1.2e-05 (time constant of 1 sec)

Table 6.5: Adaptive RED Parameters for large N simulation

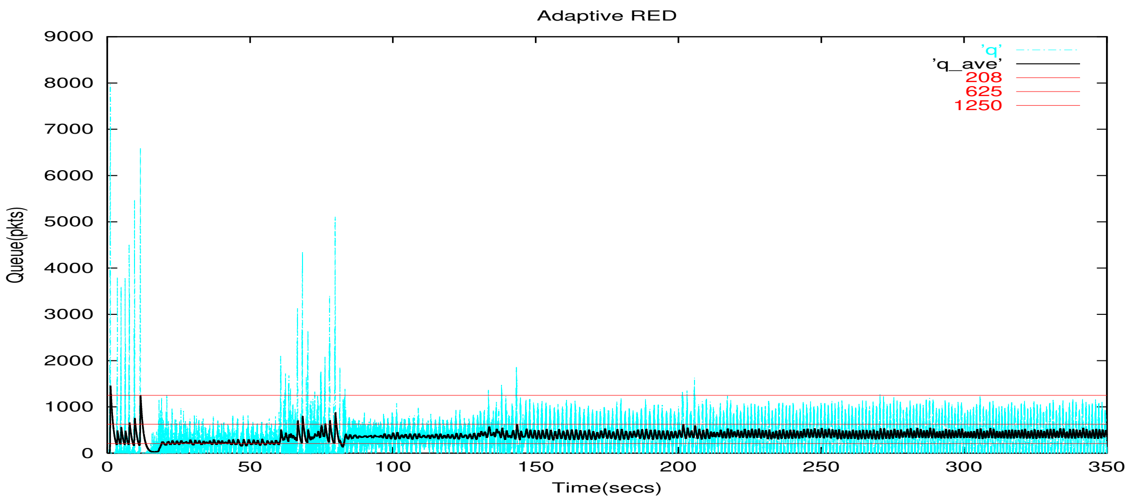


Figure 6-5: Adaptive RED Queue behavior with large N

As in the classic RED algorithm, Adaptive RED, in maintaining the queue between fixed thresholds is able to keep queues very small, resulting in low queuing delay. Figure 6-6 shows the adaptation of Pmax in response to the distance between the average queue size and a target region that lies midway between Bmin and Bmax.

SIM 2: Adaptive RED (cont'd)

Figure 6-6: Adaptive RED plot: 1/Pmax

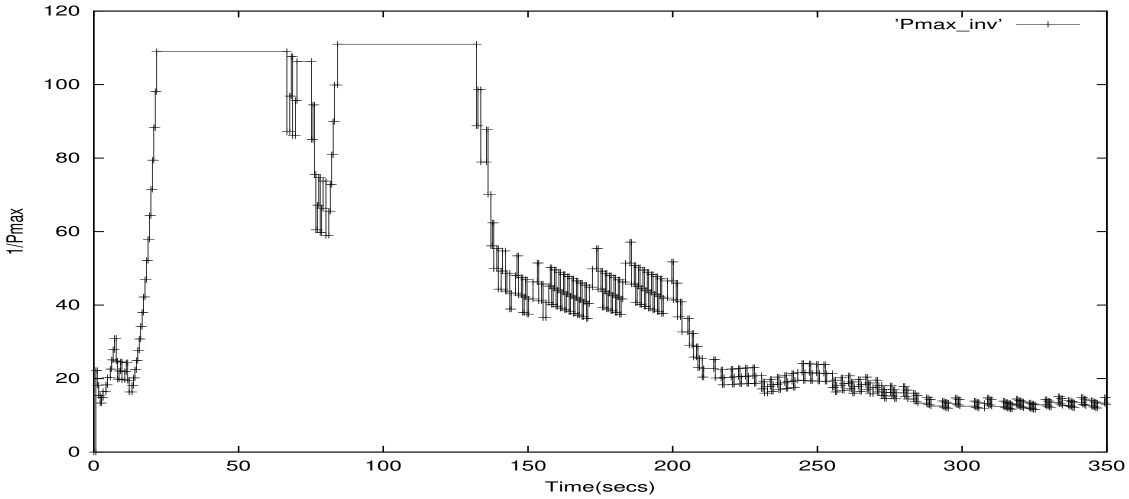
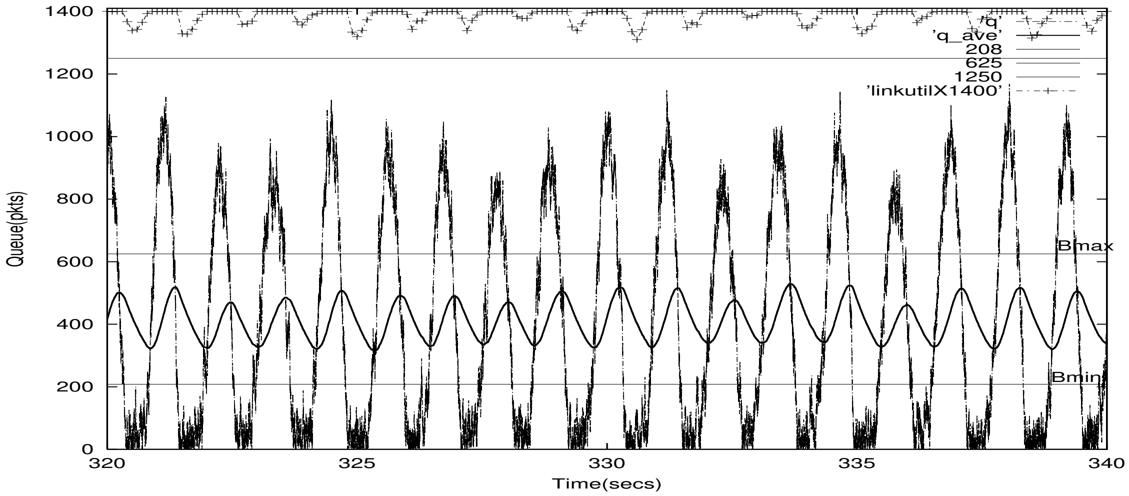


Figure 6-7: Adaptive RED, A closer look at queue size



6.3.3 SIM 2: FPQ-RED

Parameter	Value
Bmin	5.0 pkts
Bmax	Variable
Pmax	Variable
gain	1.2e-05 (time constant of 1 sec)
Blimit	83333 pkts
guessed RTT	100ms
Adjust Pdrop interval	4ms-11ms

Table 6.6: FPQ-RED Parameters for large N simulation. *Blimit*, the bottleneck buffer, is purposely set high to accommodate increased buffer size settings. *Adjust Pdrop interval* indicates the frequency with which N is recomputed (by examining the bit-vector) and Bmax and Pmax are reset accordingly.

Contrary to Adaptive RED, the first observation when assessing the performance of FPQ with specialization to RED is the very large queue size (See Figure 6-8). The plots in Figure 6-10 show how FPQ-RED adjusts Pmax and Bmax as N grows. As N increases, the dropping probability also increases. The change in Bmax is more interesting because it captures a defining characteristic of FPQ. As reflected in Equation 3.4, the target queue size that is also used to set Bmax in FPQ first shrinks as N increases. However, the target is limited to ensure that each connection has an average of six packet buffers. Thus, there is a point where the target queue, and hence Bmax will begin to increase (See Section 4.3.4). In this manner, FPQ is able avoid the timeouts that result in unpredictable and unfair connection delays. Delays in FPQ are incurred from queuing rather than from timeouts, and are evenly distributed.

Note however that in all the simulations we consider, the number of flows is kept within bounds that do not incur excessive timeouts as a result of insufficient storage buffers on intermediate links and queues.

SIM 2: FPQ-RED (cont'd)

Figure 6-8: FPQ-RED Queue behavior with large N

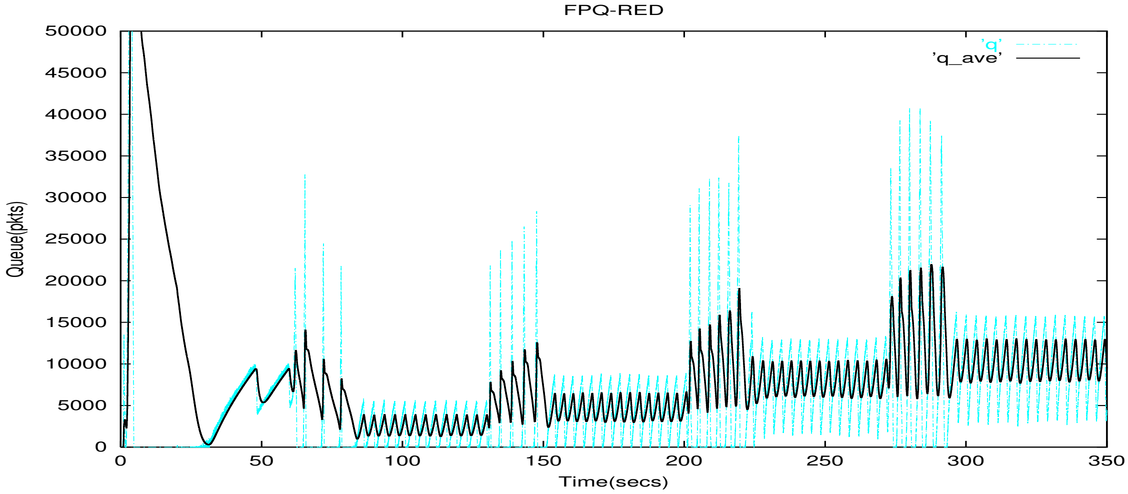
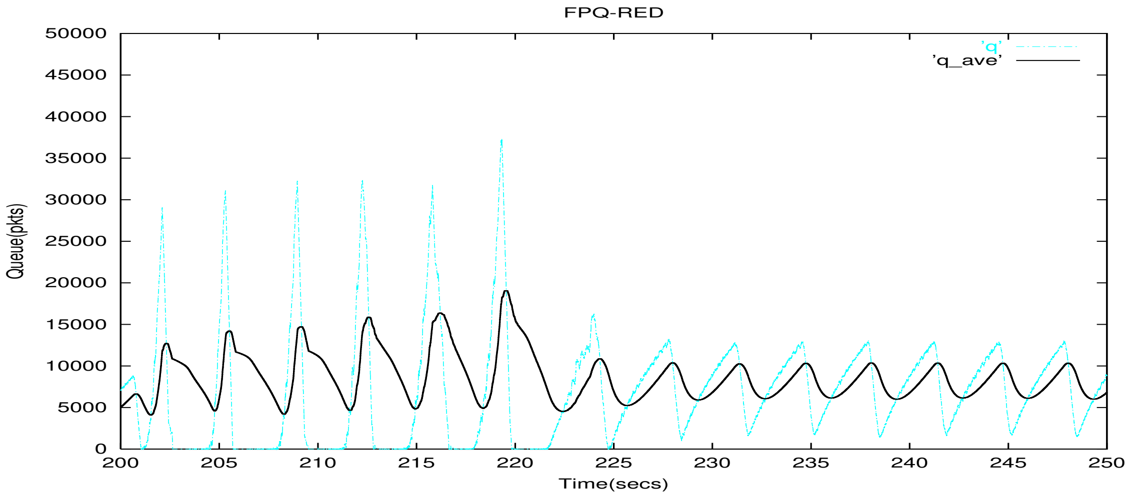


Figure 6-9: FPQ-RED, A closer look at queue size



SIM 2: FPQ-RED (cont'd)

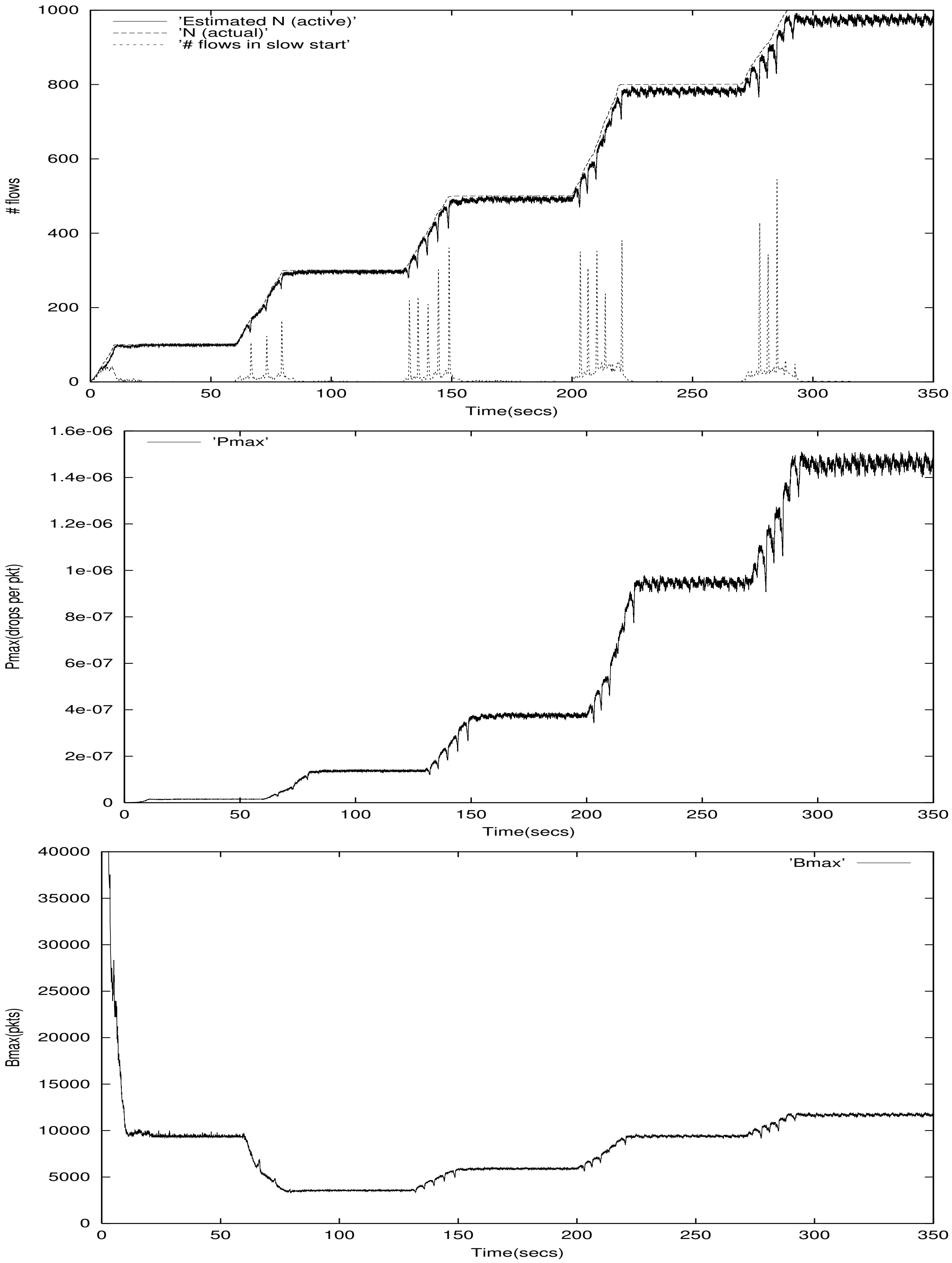


Figure 6-10: FPQ-RED plots of the number of flows, and the adaptive values of Pmax and Bmax.

6.3.4 SIM 2: Performance

Because Simulation 2 considers long-running uni-directional TCPs, performance is excellent for all schemes. 2RegionRED and FPQ-RED experience 100% link utilization because there is a sustained persistent queue. Adaptive RED keeps the queue low and link utilization as shown in Figure 6-7 averages roughly at 97%. Because this simulation concerns long-running TCPs, the goodput for all three schemes are indistinguishable and nearly optimal.

6.4 Adding Complexity to the Simulation

The last simulation we consider in this chapter demonstrates how 2RegionRED operates in a more complex traffic scenario. It should reflect more closely some aspects of the traffic behavior experienced in the real Internet. The complexity arises from 1) more frequent slow-starts that result from the presence of many short-lived flows, and from starting and stopping flows, and 2) the dynamics of two-way traffic. These are described below, and apply to Simulation 3.

Two-way Traffic Reverse traffic is added on the link.

Log-normal transfer sizes This is suggested in a paper by Floyd and Paxson [17]. This means the log of the sizes will fit a Gaussian distribution. We implement a certain number of sources can be active at any time. Each transfer (in bytes) is taken from a log-normal distribution where the normal curve is characterized by a mean of 7.0 and a standard deviation of 3.0.

Think-Times/Restarting flows When one connection completes, another connection begins after a *think-time*. This flow is considered to be a new and distinct flow in this chapter. In these simulations, the think-time is a randomly chosen time between 0 and 100s. For a more complex scenario, the think-times can be taken from a Pareto distribution characterized by a particular mean time and a tail index.

SIMULATION 3

Topology: Single bottleneck, 155Mbps

Scenario: Two-way traffic, RTTs (60ms-140ms),

Log-normal transfer sizes, Restarting flows after a think-time.

$Direction\ bn0 \rightarrow bn1(A \rightarrow B)$

050-100s: +50 flows ($N_{A2B} = 50$)

180-200s: +50 flows ($N_{A2B} = 100$)

250-300s: +50 flows ($N_{A2B} = 150$)

$Direction\ bn1 \rightarrow bn0(B \rightarrow A)$

000-010s: +10 flows ($N_{B2A} = 10$)

130-135s: +20 flows ($N_{B2A} = 30$)

230-270s: +40 flows ($N_{B2A} = 70$)

Simulation length = 320s.

These are flows of fixed size and will be starting, stopping, and restarting at variable times. Figure 6-20 shows for example, the number of active flows present in both directions at any given time for the 2RegionRED simulation. Notice that there are about half as many flows in the reverse direction as in the forward direction. The following pages show the behavior

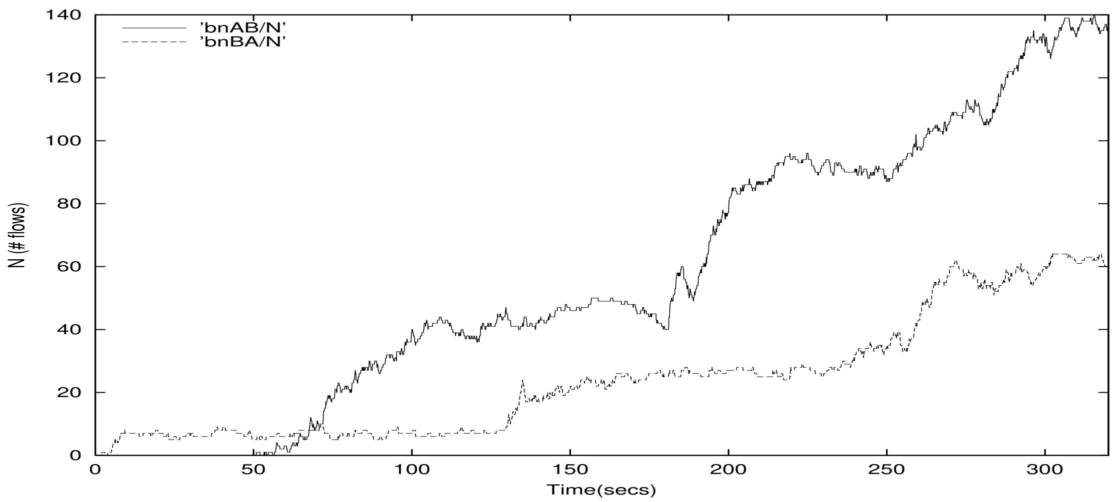


Figure 6-11: 2RegionRED: Actual N

in the queue of 2RegionRED, Adaptive RED, and FPQ-RED for Simulation 3.

6.4.1 SIM 3: 2RegionRED

Parameter	Value
Bmin	89 pkts ($N_{Bmin} = 10$)
Bflat	134 pkts
Bmax	645 pkts
Btarget	234 pkts (Bflat + 100)
Blimit	1937 pkts (1.5*DBP)
guessed RTT	100ms
PPD initial	1291 pkts (DBP)
ABSMIN interval	100ms
Adjust Pdrop interval	96ms-144ms

Table 6.7: 2RegionRED Parameters for Complex Simulation. The *Adjust Pdrop interval* indicates how frequently the high-N algorithm in Figure 4-7 is called, which adjusts the drop rate. *PPD initial* is the initial setting of the drop rate, manifested in Packets Per Drop, when the estimated queue q_est first crosses Bflat. The ABSMIN interval uses a subsampling granularity of 6.7ms (See Section 5.2.3).

The plots in Simulation 3 differ greatly from those in Simulation 2. The first thing to notice, with the addition of short-lived flows and reverse traffic, is the rapid fluctuations in queue size that result from the everchanging environment and perhaps from the effects of ACK-compression. The queue sizes also grow very high as a result of TCP slow starts.

For Simulation 2, the difficulties discussed in Section 6.3.1 (concerning the consequences of the lag in ABSMIN and an initial drop rate that is too low when the Bflat boundary is first crossed) resulted in spikes in the queue size that were infrequent and therefore easily dismissed. Here however, frequent slow starts cause these spikes to dominate the activity in the queue.

6.4.2 Difficulties with Slow Starting Flows

Slow starts can have two consequences:

1. Slow starts can worsen the consequence of the lag in ABSMIN, causing buffers to overflow before the algorithm can respond.

2. Slow starts can fool the algorithm. Firstly, because slow starts often last several round-trip times, the ABSMIN queue estimation scheme recognizes correctly that the queue is persistent. Secondly, the adaptive nature of 2RegionRED is based upon equations derived from TCP congestion avoidance behavior. Because slow starts are characterized by an exponential increase in queue size rather than a linear increase, 2RegionRED in such a circumstance infers that there are many more flows than there actually are and imposes a drop rate that is too high for the actual number of flows. This causes the queue to drain and leads to underutilization of the link as is evident in Figure 6-14. As often occurs, each time the queue drains "improperly" in this manner (for instance, even though the number of flows is large), when the queue rises again it often appears as a surge in the queue. When the queue first surges above Bflat, the effects of (1) possibly coupled by (2) may cause this behavior to repeat itself.

In the previous 2RegionRED simulations, Bmax was set to the buffer limit. In those cases, each time many new flows were introduced the queue would grow beyond the buffer limit and enter drop-everything mode which is undesirable. In this simulation we set Bmax to half the pipesize, and implement a basic form of the "gentle" algorithm. From Bmax (645 pkts) to 2*Bmax (1291 pkts), the drop rate increases linearly form 0.1 to 1.0.

When there are fewer flows as in the $B \rightarrow A$ direction, observe that 2RegionRED has greater difficulty in coping with activity such as slow starts and ACK-compression. When there are many competing flows (in the rightmost portion of Figure 6-12), a higher drop rate is required by 2RegionRED to keep the queue steady. This higher drop rate also allows slow starts to be more quickly absorbed.

SIM 3: 2RegionRED (cont'd)

Figure 6-12: 2RegionRED ($A \rightarrow B$): Entire simulation

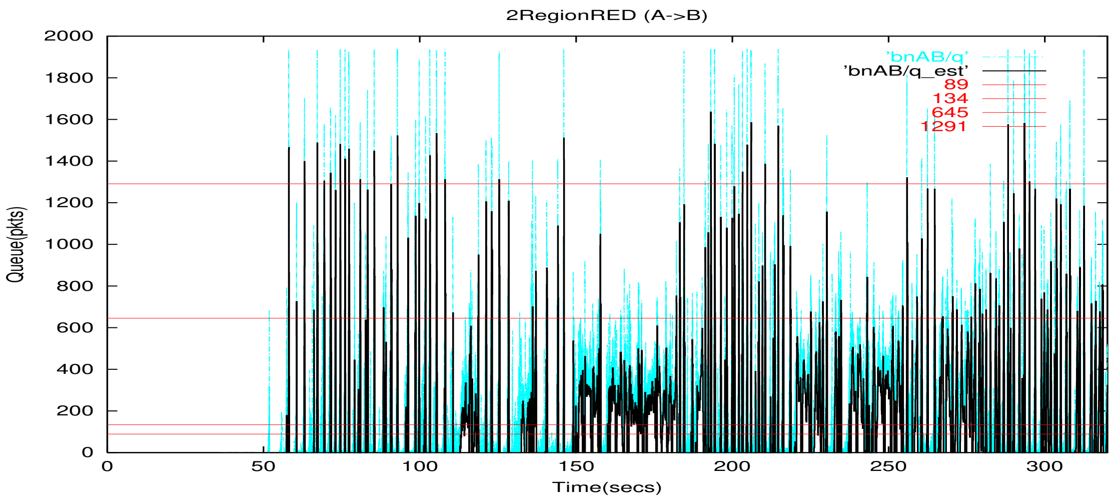
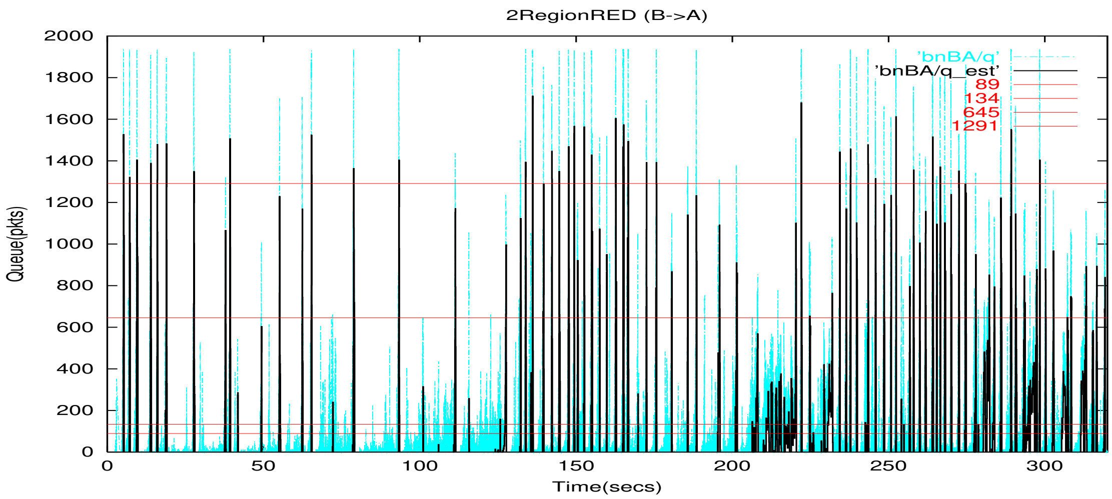


Figure 6-13: 2RegionRED ($B \rightarrow A$): Entire simulation



SIM 3: 2RegionRED (cont'd)

Figure 6-14: 2RegionRED underutilized link

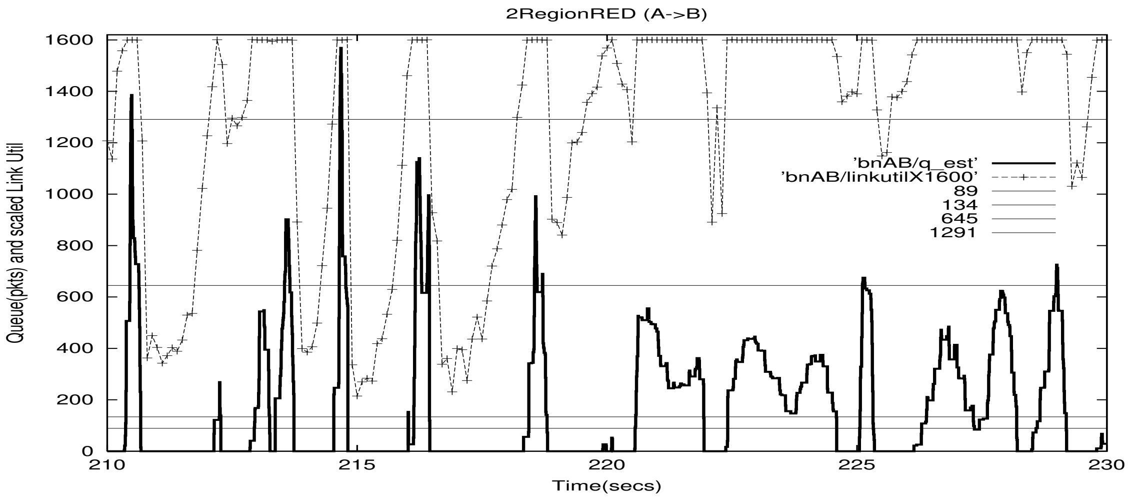
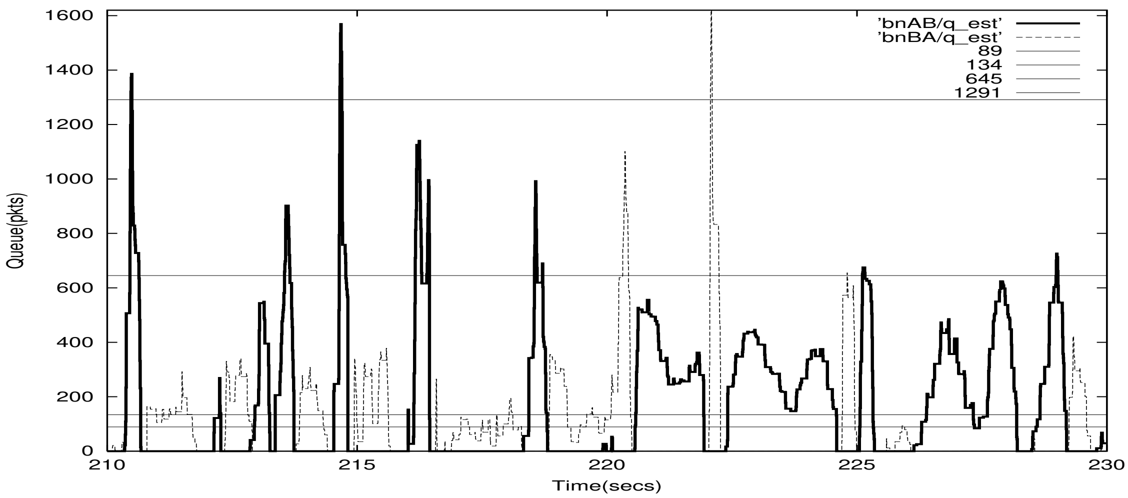


Figure 6-15: 2RegionRED alternating queues



6.4.3 SIM 3: Adaptive RED

Parameter	Value
Bmin	32.3 pkts
Bmax	96.9 pkts
2*Bmax	193.8 pkts
Pmax	Variable
Blimit	1937 pkts (1.5*DBP)
gain	7.74e-05 (time constant of 1 second)

Table 6.8: Adaptive RED Parameters for complex simulation

Figure 6-16: Adaptive RED ($A \rightarrow B$): Entire simulation

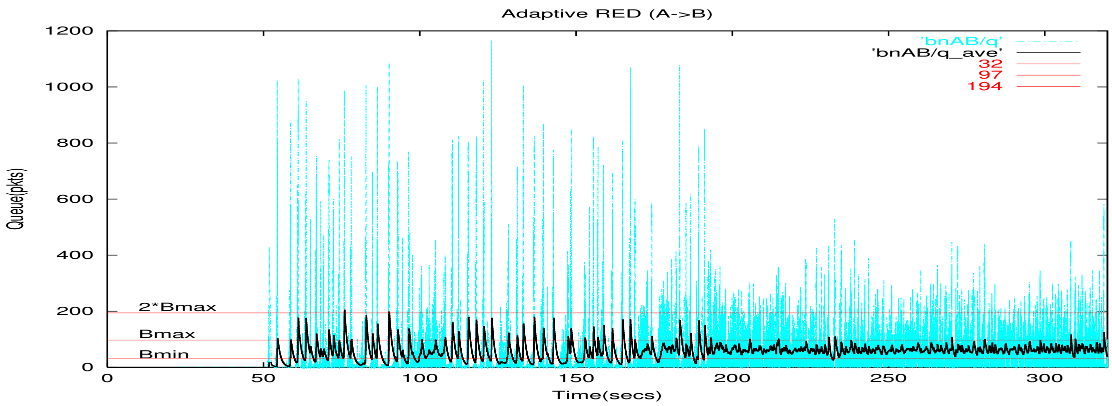
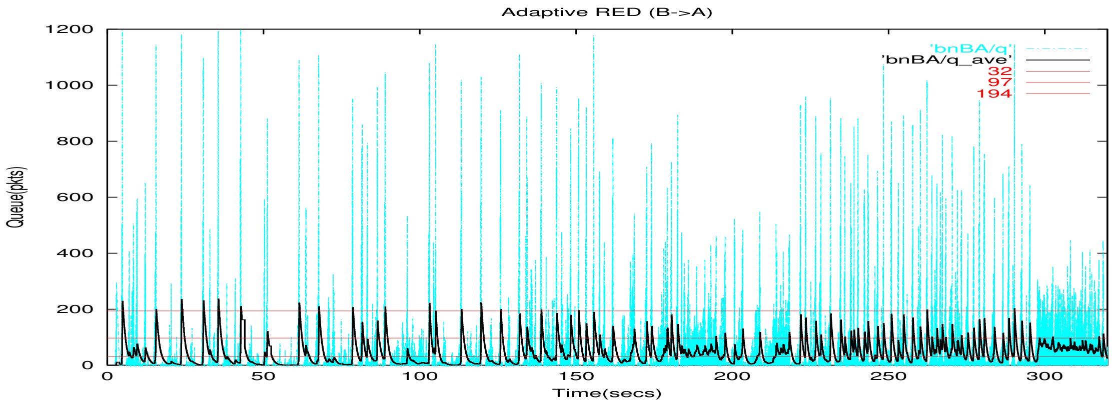


Figure 6-17: Adaptive RED ($B \rightarrow A$): Entire simulation



6.4.4 SIM 3: FPQ-RED

Parameter	Value
Bmin	5.0 pkts
Bmax	Variable
Pmax	Variable
gain	7.74e-05 (time constant of 1 sec)
Blimit	19375 pkts (15 * DBP)
guessed RTT	100ms
Adjust Pdrop interval	4ms-11ms

Table 6.9: FPQ-RED Parameters for large N simulation. *Blimit*, the bottleneck buffer, is purposely set high. *Adjust Pdrop interval* indicates the frequency with which N is recomputed (by examining the bit-vector) and Bmax and Pmax are reset accordingly.

Figure 6-18: FPQ-RED $(A \rightarrow B)$: Entire simulation

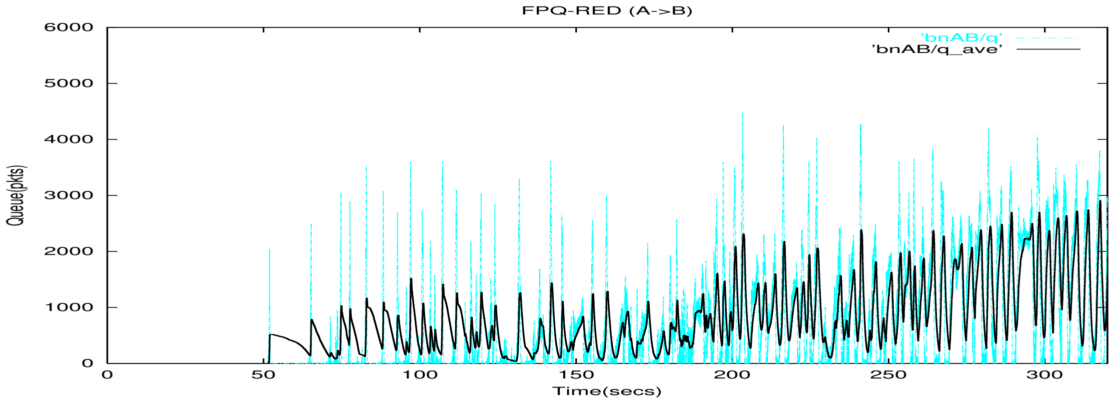
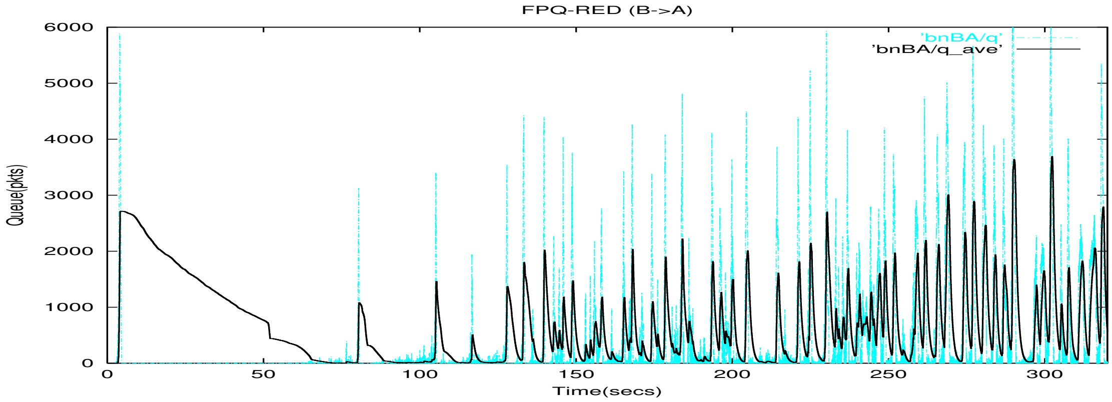


Figure 6-19: FPQ-RED $(B \rightarrow A)$: Entire simulation



6.4.5 SIM 3: Performance

None of the algorithms show terrific performance in this more realistic scenario characterized by frequent short-lived, slow-starting flows. For this particular simulation, average link utilization results for the entire simulation are revealed in Table 6.10. Furthermore, scatterplots of goodput versus filesize for 2RegionRED and Adaptive RED reveal loosely comparable results. Simulations with a constant number of flows or simulations that plot goodput in a 3-Dimensional form vs. time may be more revealing for future simulation studies.

Algorithm	$A \rightarrow B$ (%)	$B \rightarrow A$ (%)
2RegionRED	58.5	50.7
Adaptive RED	56.5	40.6
FPQ-RED	61.3	46.6

Table 6.10: Comparison of Link Utilization. What this table establishes is that none of the algorithms work very well in sustaining high link utilization. In general, link utilization is mediocre.

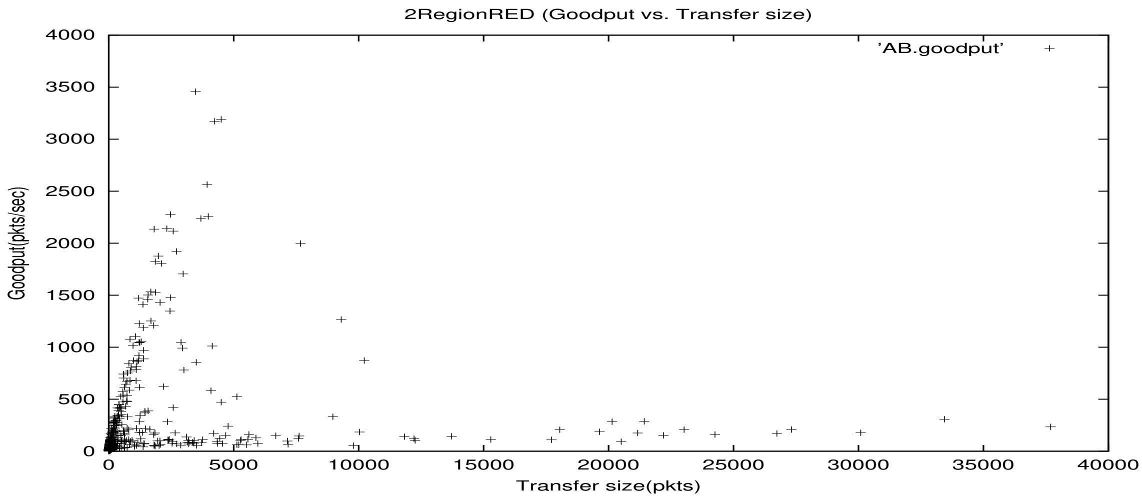


Figure 6-20: 2RegionRED: Scatterplot of Goodput vs. Transfer Size. This plot gives a sense of the way in which lognormal transfer sizes are distributed in the simulation, characterized by a greater density of shorter flows. Points are only shown for flows that have completed. Because flows are introduced into the system at different times, flows starting later are competing with more TCPs. Keep this in mind when observing the distribution of the goodput for any particular transfer size.

Chapter 7

Conclusions and Future Work

7.1 Summary

The two main contributions of this thesis are

1. the design and intuition behind the 2RegionRED congestion control algorithm
2. the analysis of different mechanisms for measuring the persistent queue

Persistent Queue One contribution of this thesis involved exploring different mechanisms for measuring the persistent queue, under the assumption that buffers exist to absorb transient bursts. Most of the algorithms we have studied do not pay much attention to the method of queue estimation. We discover that ABSMIN is superior to EWMA and EWMA' in measuring the persistent queue. Its primary drawback is a result of the lag– we cannot know that a queue is persistent rather than a transient until after the fact.

Two Regions 2RegionRED is an adaptive algorithm which uses the number N of competing TCPs as its measure of congestion. The design of 2RegionRED is influenced by the new RED signal (described in Section 4.1.2) which emphasizes sustaining as small a queue necessary that still permits full link utilization. For any N, 2RegionRED enforces a drop rate that relies on a relationship between the drop rate imposed, the level of multiplexing, and the average TCP window size.

In order to enforce the proper drop rate, the intuition behind 2RegionRED is that depending on N, a required drop rate of less than or greater than one drop per RTT calls for two different mechanisms (applied in the Low-N and High-N regions respectively).

This notion of separating the required drop rate into two different regions is perhaps the most significant point of this work. Other algorithms may behave accordingly without realizing it, for example by implementing a function that increases the drop rate gradually across both regions as the measure of congestion increases. However it is in this thesis that we explicitly make note of this. The 2RegionRED algorithm and implementation we have presented takes advantage of this distinction between the two regions.

Adaptive, Free of Tuning Knobs As a way to enhance the robustness of a congestion control algorithm, 2RegionRED is also adaptive in two manners.

1. A queue threshold Bflat separates the Low-N and High-N regions of control. In the Low-N Region, Figure 6-1 illustrated that as N increases, a deterministic drop at Bmin is sufficient to accomplish what classic RED does probabilistically.

2. In the High-N Region, the drop rate adapts by making an estimate of N from looking at correlations between an imposed drop rate and the resulting change in the queue size.

We have described, for 2RegionRED, an intuitive and logical way to initialize its parameters. Because 2RegionRED is self-adapting to changing loads, the need for manual resetting of parameters is eliminated. Even if one wanted to change the level of Bmin, for example, for a different tradeoff (See Section 4.3.1), there is a very logical method of choosing the desired setting.

The only other parameter that embodies some uncertainty is the assumed average RTT. The RTT is used in the equations which compute the proper drop rate. Its value also affects the accuracy of the ABSMIN queue estimation method. One of the original goals of 2RegionRED was to design an algorithm that does not require any knowledge of round-trip time. However, even in the Slope-Controlled RED proposition we found this to be unavoidable. Purposely avoiding the route of using informed TCP protocols (that is, embedding connection-specific information regarding window size or RTT, etc., inside headers to be made accessible to the router), 2RegionRED takes the common approach of assuming a canonical RTT of 100ms. If it can be shown that network performance would benefit significantly then it may be worthwhile to adjust the value of the canonical RTT. However, given the complicated and transient dynamics in the Internet, not to mention the complexity introduced when considering multiple bottlenecks, we do not believe that such fine-tuning will be necessary.

Difficulties As seen in the simulations, 2RegionRED has two main difficulties. The first is coping with behavior that deviates significantly from that exhibited by TCP in congestion avoidance, such as slow starts and ACK-compression. Because the algorithm is designed assuming TCPs are in steady-state, when the queue grows at an exponential rate, the algorithm calculates that there are more flows than actual and applies a drop rate that may be too high, causing queues to drain and links to be underutilized.

Secondly, these problems are aggravated because of the extra round trips that the algorithm takes before the dropping probability stabilizes around the desired value. 2RegionRED is prevented from reaching the appropriate drop rate sooner because of the following:

1. The lag on the growing queue when estimating the persistent queue using ABSMIN
2. Too small a drop rate when crossing from the Low-N to High-N Regions
3. The coarse granularity (roughly once every round trip) at which adjustments in the dropping probability are made.

2RegionRED is more sensitive to these when there are a small number of flows.

Another interesting observation is that in the Low-N algorithm, imposing the single deterministic drop that is directly linked to a particular queue value may lead to link underutilization. This can happen as a result of a highly fluctuating queue caused by ACK-compression as well as slow starts, especially when there are few flows.

All of these problems are rather fundamental difficulties of 2RegionRED that need to be dealt with before we can expect 2RegionRED to operate as smoothly as designed.

7.2 Future Work

As for improving 2RegionRED in any of the aspects mentioned earlier, one might investigate either ways to eliminate difficult behavior, or ways in which to cope or take advantage of it. For instance, it may be worth investigating whether slow starting flows really should be treated differently.

Although 2RegionRED's measurement of load is decoupled from the queue size, the algorithm still relies on an accurate measure of the queue because it computes and responds to changes in the queue. This applies to the preliminary proposition, Slope-Controlled RED, as well (See Section 4.2.1). Not surprisingly, both 2RegionRED and Slope-Controlled RED suffer in performance when other phenomena such as slow starts and ACK-compression arise. This prompts us to think twice about whether using mechanisms that are linked to particular queue sizes or changes in queue size can be relied upon, given the possibility that there may be unanticipated queue behaviors currently not understood. This stresses the need for a comprehensive understanding of the complex and real dynamics that may arise in the Internet.

With regards to queue estimation, it is interesting to see if there is a way to avoid the negative consequences of the lag in measuring the persistent queue, especially on a rising queue.

We saw in the previous chapter that there are also other algorithms that do not require sophisticated tuning. For example, Adaptive RED appeared to operate well while surprisingly able to keep the queue very small. A more in-depth study of adaptive algorithms and simulations under a variety of different scenarios is warranted.

Dependence upon TCP congestion control dynamics 2RegionRED was formulated assuming that the traditional AIMD TCP is and will remain the dominant transport protocol used. An interesting philosophical question that is worth exploring is how dependent the congestion control algorithm should be upon TCP dynamics. 2RegionRED, along with other algorithms (such as FPQ) that use the number of flows as a measure of load formulate their drop rate based upon TCP's congestion control dynamics. They assume that a well-behaved TCP will halve its window on the receipt of a congestion notification or a drop, and can infer how this will impact the queue size. However, what if TCP dynamics were to change? For instance, variations on TCP's congestion control may be used by TCP-friendly applications that are not AIMD with the standard increase factor of one and decrease factor of two [5]. Furthermore, an increasing number of applications in the Internet do not use TCP as their underlying transport protocol, for good reasons. Applications such as streaming audio and video are not well suited to TCP's enforcement of reliable in-order delivery. Instead, these applications use custom protocols which run over the User Datagram Protocol (UDP). Finally, the end-to-end arguments [30] suggest that the core should remain fixed while the end points be given more flexibility to change. With this in mind, what are the essential principles by which a congestion control algorithm should be built? What mechanisms are viable?

Chapter 8

Tables

N	TPUT	PPD	DPP	RPD	DPR	W(Optimal Buffer)
1	1.55e+08	2.5e+06	3.99e-07	1.29e+03	0.000774	1.29e+03
10	1.55e+07	1.2e+04	8.31e-05	9.01	0.111	89.1
20	7.75e+06	2.94e+03	0.00034	2.24	0.447	43.8
30	5.17e+06	1.31e+03	0.000765	1	0.999	29
75	2.07e+06	217	0.00461	0.167	5.98	11.5
100	1.55e+06	125	0.008	0.0964	10.4	8.64
200	7.75e+05	34.4	0.0291	0.0266	37.6	4.31
300	5.17e+05	16.7	0.0599	0.0129	77.4	2.87
400	3.88e+05	10.2	0.0981	0.00789	127	2.15
500	3.1e+05	7.04	0.142	0.00545	184	1.72
1000	1.55e+05	2.41	0.416	0.00186	537	0.861

Table 8.1: Choosing N_{Bmin}: 155 Mbps, 100ms RTT

N	TPUT	PPD	DPP	RPD	DPR	W(Optimal Buffer)
1	6.22e+08	4.03e+07	2.48e-08	5.18e+03	0.000193	5.18e+03
10	6.22e+07	1.92e+05	5.2e-06	35.8	0.0279	357
20	3.11e+07	4.66e+04	2.15e-05	8.84	0.113	176
40	1.56e+07	1.15e+04	8.68e-05	2.2	0.454	87.1
60	1.04e+07	5.12e+03	0.000195	0.982	1.02	57.9
75	8.29e+06	3.28e+03	0.000305	0.63	1.59	46.3
100	6.22e+06	1.86e+03	0.000539	0.357	2.8	34.7
200	3.11e+06	475	0.0021	0.0915	10.9	17.3
300	2.07e+06	217	0.00461	0.0418	23.9	11.5
400	1.56e+06	125	0.00799	0.0241	41.5	8.65
500	1.24e+06	82.1	0.0122	0.0158	63.2	6.92
1000	6.22e+05	23.1	0.0433	0.00446	224	3.46

Table 8.2: Choosing N_{Bmin}: 622 Mbps, 100ms RTT

N	TPUT	PPD	DPP	RPD	DPR	W(Optimal Buffer)
1	1e+09	1.04e+08	9.6e-09	8.33e+03	0.00012	8.33e+03
2	5e+08	1.67e+07	6e-08	1.67e+03	0.0006	3.33e+03
5	2e+08	2.13e+06	4.7e-07	238	0.0042	1.19e+03
10	1e+08	4.96e+05	2.01e-06	57.6	0.0174	575
40	2.5e+07	2.96e+04	3.37e-05	3.53	0.284	140
75	1.33e+07	8.42e+03	0.000119	1.01	0.995	74.4
100	1e+07	4.74e+03	0.000211	0.567	1.76	55.7
200	5e+06	1.2e+03	0.000831	0.144	6.94	27.8
500	2e+06	202	0.00495	0.0242	41.3	11.1
1000	1e+06	54.7	0.0183	0.00656	152	5.56
1500	6.67e+05	26.1	0.0383	0.00314	319	3.7
2000	5e+05	15.7	0.0635	0.00189	529	2.78

Table 8.3: Choosing N_{Bmin}: 1 Gbps link, 100ms RTT

Bibliography

[1] M. Allman, V. Paxson, and W. Stevens. RFC2581: TCP Congestion Control, April 1999.

[2] S. Athuraliya, S. Low, and D. Lapsley. Random early marking. In *First International Workshop on Quality of future Internet Services (QofIS'2000)*, Berlin, Germany, September 2000.

[3] J. Aweya, M. Ouellette, D. Y. Montuno, and A. Chapman. Enhancing TCP performance with a load-adaptive RED mechanism. In *International Journal of Network Management 2001*, volume 11, pages 31–50.

[4] B. Braden, et al. RFC2309: Recommendations on Queue Management and Congestion Avoidance in the Internet, April 1998.

[5] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proceedings of INFOCOM 2001*, April 2001.

[6] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. Tuning RED for Web Traffic. In *Proc. of ACM SIGCOMM 2000*, pages 139–150, Stockholm, Sweden, August-September 2000.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. In *Proceedings of INFOCOM '99*, March 1999.

[8] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A New Class of Active Queue Management Algorithms, CSE-TR-387-99. Technical report, U. Michigan, 1999.

[9] S. Floyd. Note of Aug 14, 1999 in the red-impl mailing list archives.

[10] S. Floyd. Notes on RED gentle_ tests in NS.

[11] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[12] S. Floyd. (email) Optimal functions for computing the drop probability, October 1997.

[13] S. Floyd. (email) RED: Discussions of Setting Parameters, November 1997.

[14] S. Floyd. A Report on Some Recent Developments in TCP Congestion Control, in submission, June 2000.

[15] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, August 1, 2001, under submission.

[16] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, volume 1, pages 397–413, August 1993.

[17] S. Floyd and V. Paxson. Difficulties in Simulating the Internet.

[18] C.V. Hollot, V. Misra, D. Towsley, and W. Gong. A Control Theoretic Analysis of RED. *To appear in IEEE Infocom 2001*, April 2001.

[19] V. Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review*, 18(4):314–329, August 1988.

[20] V. Jacobson, K. Nichols, and K. Poduri. RED in a Different Light, Draft of Sept. 30, 1999.

[21] C. Lefelhocz, B. Lyles, S. Shenker, and L. Zhang. Congestion Control for Best Effort Service: Why We Need a New Paradigm. *IEEE Network*, 10(1), January/February 1996.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[23] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. Technical report, June 1999.

[24] M. May, T. Bonald, and J. Bolot. Analytic Evaluation of RED Performance. *Presented at Infocom, 2000.*

[25] S. McCanne and S. Floyd. ns Network Simulator.

[26] R. T. Morris. *Scalable TCP Congestion Control.* PhD dissertation, Harvard University, January 1999.

[27] R. T. Morris. Scalable TCP Congestion Control. In *IEEE INFOCOM 2000*, pages 1176–1183, Tel Aviv, March 2000.

[28] T. J. Ott, T. V. Lakshman, and L.H. Wong. SRED: Stabilized RED. In *Proceedings of IEEE/INFOCOM*, 1999.

[29] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach.* Morgan Kaufmann Publishers, Inc., San Francisco, CA., 1996.

[30] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, Nov 1984.

[31] C. Q. Yang and A. V. S. Reddy. A Taxonomy for Congestion Control Algorithms in Packet Switching Networks. *IEEE Network Magazine*, 9(5), July/August 1995.

[32] L. Zhang, S. Shenker, and D.D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proc. of ACM SIGCOMM 1991*, pages 133–147, September 1991.

[33] Y. Zhang and L. Qiu. Understanding the End-to-End Performance Impact of RED in a Heterogeneous Environment, CS Technical Report 2000-1802. Technical report, Cornell, July 2000.