A Loss and Queuing-Delay Controller for Router Buffer Management

Long Le Kevin Jeffay F. Donelson Smith
Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/Research/dirt

Abstract — Active queue management (AQM) in routers has been proposed as a solution to some of the scalability issues associated with TCP's pure end-to-end approach to congestion control. However, beyond congestion control, controlling queues in routers is important because unstable router queues can cause poor application performance. Existing AQM schemes explicitly try to control router queues by probabilistically dropping (or marking) packets. We argue that while controlling router queues is important, this control needs to be tempered by a consideration of the overall loss-rate at the router. Solely attempting to control queue length can induce loss-rates that have as negative an effect on application and network performance as the large queues that existing AQM schemes were trying to avoid. Thus controlling queue length without regard to loss-rate can be counterproductive. In this work we demonstrate that by jointly controlling queue length and loss-rate, both network and application performance are improved. We present a novel AQM design that attempts to simultaneously optimize queue length and loss-rate. Our algorithm, called *loss and queuing delay control* (LQD), is a control theoretic scheme that explicitly treats loss-rate as a control parameter. LQD is shown to provide stable control analytically and is evaluated empirically by comparing its performance against other control theoretic AQM designs (PI and REM). The results of evaluation in a laboratory testbed under realistic traffic mixes and loads show that LQD results in lower overall loss rates and that applications see lower average queue lengths than with PI or REM.

1 Introduction

Congestion control on the Internet has historically been performed end-to-end with end-systems assuming the responsibility for detecting congestion and reacting to it appropriately. Current TCP implementations detect instances of packet loss, interpret these events as indicators of congestion, and reduce the rate at which they are transmitting data by reducing the connection's window size. This congestion reaction (combined with a linear probing congestion avoidance mechanism) successfully eliminated the occurrence of congestion collapse events on the Internet and has enabled the growth of the Internet to its current size. Nonetheless, concerns have been raised about the future of pure end-to-end approaches to congestion control [2, 5]. In response to these concerns, router-based congestion control schemes known as active queue management (AQM) have been developed and proposed for deployment on the Internet [2]. With AQM, it is now possible for end-systems to receive a signal of incipient congestion prior to the actual occurrence of congestion. The signal can be implicit, realized by a router dropping a packet from a connection even though resources exist to enqueue and forward the packet, or explicit, realized by the routers' AQM scheme setting an explicit congestion notification (ECN) bit in the packet's header and forwarding the packet.

Between the methods of dropping and ECN marking of packets, ECN marking and forwarding is clearly preferred. Indeed a previous study of marking versus dropping showed that when combined with prominent AQM schemes, ECN allowed interactive applications to experience significantly reduced response times for request-response exchanges, and allowed service providers to obtain higher link utilization and lower loss rates [16].

The positive results of ECN, however, are confounded by the lack of ECN deployment in the current Internet. For example, a recent study (2004) showed that only 2.1% of web servers on the Internet had correctly deployed ECN [19]. Without ECN enabled at the end systems, dropping packets is the only mechanism AQM schemes can use to signal incipient congestion. Ultimately, the goal of AQM is to ensure queues never overflow (*i.e.*, ensure that true congestion does not occur). AQM schemes typically avoid queue overflows by dropping packets aggressively when a router's queue grows larger than a certain threshold. In this paper, we argue that while it is important to control router queues, this control should not be performed without regard to the resulting loss-rate. Solely attempting to control queue length can induce loss-rates that have as negative an effect on application and network performance as the large queues that existing AQM schemes are trying to avoid. Thus controlling queue length without regard to loss-rate can be counterproductive.

We propose a new AQM scheme that controls both loss rate and queuing delay at a router. Our algorithm, *loss and queuing delay control* (LQD), dynamically balances loss rate and queuing delay at a router to improve network and application performance. LQD is a control theoretic scheme that explicitly treats loss-rate as a control parameter (in addition to a target queue length parameter). LQD is shown to provide stable control analytically and empirically. In the latter case, LQD is evaluated in a large-scale laboratory network testbed across a range of realistic workloads. The workloads are derived from measurements on Abilene (*i.e.*, Internet2)

Published in: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems,
Lisbon, Portugal, July 2006, *to appear*.

and the UNC campus. Application and network performance are compared for LQD and several prominent AQM schemes of similar design (PI and REM). The results show that LQD results in lower overall loss rates and that applications see lower average queue lengths than with PI or REM.

The rest of the paper is structured as follows. Section 2 discusses previous related work. Section 3 presents our LQD scheme. Section 4 explains our experimental evaluation methodology and Section 5 presents the results of an extensive performance study where our LQD scheme is compared to several prominent AQM schemes from the literature. Section 6 concludes our paper.

2 Background and Related Work

The original AQM design, called *random early detection* (RED), used a weighted-average queue length as a measure of congestion [8]. When this weighted average is smaller than a minimum threshold (min_{th}), no packets are marked or dropped. When the average queue length is between the minimum threshold and the maximum threshold (max_{th}), the probability of marking or dropping packets varies linearly between 0 and a maximum drop probability (max_p, typically 0.10). If the average queue length exceeds max_{th}, all packets are dropped. (The actual size of the queue must be greater than max_{th} to absorb transient bursts of packet arrivals.)

Since the development of RED, numerous additional schemes have been developed. Of special interest are the class of designs based on the principles of control theory. In a previous study [16], and in follow-on work [15], we showed that *proportional integral* (PI) design [11] and the *random exponential marking* (REM) design [1] were the best performing AQM designs. When compared against ARED [7], AVQ [14], SFB [4], Blue [5], RIO-PS [9], and AFD [21] in our testbed, REM and PI consistently provided the best application and network performance. For this reason, we limit ourselves here to a discussion of, and later a comparison between, PI and REM.

The PI controller is based on a linear model of TCP throughput and AQM dynamics [11]. PI attempts to regulate the queue length in a router to match a target value called the "queue reference," q_{ref}. PI samples the instantaneous queue length at a constant frequency. The drop probability is computed at each sampling interval based on the current and previous queue length samples. PI control is such that the drop probability increases in sampling intervals when the queue length is higher than its target value. The drop probability also increases if the queue has grown since the last sample (reflecting an increase in network traffic). Conversely, the drop probability is reduced when the queue length is lower than its target value or the queue length has decreased since its last sample. The sampling interval and the coefficients in the control equation depend on the link capacity, the maximum RTT and the expected number of active flows using the link.

The REM controller is conceptually similar to PI. REM periodically updates a congestion measure called "price" that reflects the mismatch between packet arrival and departure rates at the link (*i.e.*, the difference between the demand and the service rate), and the queue length mismatch (*i.e.*, the difference between the actual queue length and its target value). The price measure is computed at each sampling point based on the link capacity (in packet departures per unit time), the instantaneous queue length, and the packet arrival rate. As with PI, the control target is a particular queue length. In overload situations, the congestion price increases due to the rate mismatch and the queue length mismatch. As the price measure increases, more packets are dropped or marked to signal TCP senders to reduce their transmission rate. When congestion abates, the congestion price is reduced because the mismatches are now negative. This causes REM to drop or mark fewer packets and allows the senders to potentially increase their transmission rate.

While PI and REM are a small sample of the large body of literature in AQM, it is the case that virtually all existing schemes have all focused solely on controlling a router's queue length (via different mechanisms and for different objectives). In contrast, our focus is on the joint control of queue length and loss rate. The majority of drops that occur when using AQM are "early drops" and are made when buffer capacity exists to queue the packet. We propose a new queue management scheme that considers both a reference (target) queue length and the current loss-rate when deciding whether to drop or enqueue an arriving packet.

3 The LQD Algorithm

Controlling the length of a router's queue is an important and difficult task. A large queue causes arriving packets to be subject to long queuing delays and can cause instability in the TCP control feedback loop [17]. A large queue can be prevented by dropping packets aggressively, however, a short target queue length runs the risk that the queue can drain quickly and become empty before new packets arrive. In this case, the link is underutilized and the router has unnecessarily dropped packets that could have been enqueued and forwarded without adversely affecting link congestion.

We argue that while controlling routers' queues is an important goal, it should not be achieved by simply dropping arriving packets. This issue is particularly important because of the bursty characteristics of Internet traffic that can cause *temporary congestion* at routers [3, 21]. (AQM schemes such as RED and its derivatives attempt to deal with bursty arrivals by using a low-pass filter to smooth the measure of average queue length. However, as we have previously shown, this control is ineffective [16].) We believe an AQM scheme should be flexible enough to absorb short-term bursts where the input rate *temporarily* exceeds the link capacity. On the other hand, an AQM design should be able to control routers' queue when *persistent congestion* occurs.

This design distinguishes our AQM scheme from existing AQM schemes that simply try to control routers' queue at any cost (independent of its effect on the environment). To this end, we propose a new AQM scheme called *loss and queuing delay* (LQD) controller that provides a framework for balancing loss rate and queuing delay.

3.1 Algorithm Description

Like most AQM schemes, on each packet arrival LQD computes a drop probability $p(t)$ which is used to decide whether the arriving packet is to be dropped or forwarded. Let T be the sampling interval and $l(t)$ be the estimated packet loss rate (*i.e.*, the ratio of the number of dropped packets to the number of arriving packets). The drop probability at time kT is computed as

$$p(kT) = p((k-1)T) + a \times (q(kT) - q_{ref}) \; - \; b \times (l(kT) - p_{ref})$$

where a and b are coefficients of the LQD controller and $p_{ref} \geq 0$, and $q_{ref} > 0$, are the target loss rate and target queue length respectively. The drop probability is increased when the queue length is larger than the queue target and is decreased otherwise. However, when the loss rate grows larger than its threshold, the drop probability is adjusted downward and the queue is allowed to grow temporarily. The coefficients a and b allow a router to balance between queuing delay and packet loss rate. The coefficient a specifies how large the queue can grow and the coefficient b allows the router to adjust the loss rate and absorb transient congestion. In general, a should be significantly smaller than b since the range of values for queue length (tens to hundreds) is significantly larger than the range of values for packet loss rate (hundredths to tenths). We observed from experimental data that the difference between the actual queue length and the queue reference is order of tens in dynamic environments, and the packet loss rate is order of hundredths. Based on these results of empirical analysis, in all experiments we set $a = 0.0001$ and $b = 0.1$ to balance the relative contributions to the drop probability of the queue length mismatch and the loss-rate miss match.

3.2 Stability Analysis

The basic issue with any controller is the ability to realize stable control. Here we give a brief sketch of a stability analysis useing the framework developed by Misra *et al.* [18]. The analysis necessarily makes a number of simplifying assumptions and is provided simply to provide the intuition for stability. Ultimately, our real proof of stability derives from the performance achieved with LQD in practice.

Consider a system with N TCP connections sharing a bottleneck link with capacity C. For analytic tractability, we assume that the system is homogenous. Let $w(t)$ be the congestion window and τ be the propagation delay of these connections. Let $q(t)$ and $p(t)$ be the queue length and drop probability at the bottleneck router. The evolution of an end-system's window size and the queue at a bottleneck router are given by

$$\frac{dw(t)}{dt} = f(p,q,w) = \frac{1}{\tau} - \frac{w(t)w(t-\tau)}{\eta\tau} p(t-\tau) \qquad (1)$$

$$\frac{dq(t)}{dt} = g(p,q,w) = \frac{N}{\tau} w(t) - C \qquad (2)$$

where η is the number of data segments acknowledged by an ACK (usually $\eta = 2$). The actual packet loss rate can be approximated by the drop probability and the control equation of LQD can also be formulated as

$$\frac{dp(t)}{dt} = h(p,q,w) = a(q(t) - q_{ref}) - b(p(t) - p_{ref}) \qquad (3)$$

In steady state, the system operates around an operating point (w_0, p_0) where

$$w_0 = \frac{\tau C}{N} \text{ and } p_0 = \frac{\eta}{w_0^2} = \frac{\eta N^2}{\tau^2 C^2}.$$

We can linearize equations (1), (2), and (3) around the operating point using the Taylor series approximation. Let $\partial w = w - w_0$ and $\partial p = p - p_0$. Since

$$\frac{\partial f}{\partial q} = \frac{\partial g}{\partial p} = \frac{\partial g}{\partial q} = \frac{\partial h}{\partial w} = 0, \; \frac{\partial h}{\partial q} = a, \text{ and } \frac{\partial h}{\partial p} = b,$$

we have

$$\frac{d\delta w}{dt} = \frac{\partial f}{\partial w}\delta w + \frac{\partial f}{\partial p}\delta p + \frac{\partial f}{\partial q}\delta q \qquad (4)$$

This implies

$$\frac{d\delta w}{dt} = K_{11}\delta w + K_{12}\delta w(t-\tau) + K_{13}\delta p(t-\tau) \qquad (5)$$

$$\frac{d\delta q}{dt} = \frac{\partial g}{\partial w}\delta w + \frac{\partial g}{\partial p}\delta p + \frac{\partial g}{\partial q}\delta q = K_{21}\delta w \qquad (6)$$

$$\frac{d\delta p}{dt} = \frac{\partial h}{\partial w}\delta w + \frac{\partial h}{\partial p}\delta p + \frac{\partial h}{\partial q}\delta q = a\delta q - b\delta p \qquad (7)$$

where $K_{11} = K_{12} = \frac{\partial f}{\partial w} = -\frac{2N}{C\tau^2}$, $K_{13} = \frac{\partial f}{\partial p} - \frac{\tau C^2}{\eta N^2}$, and $K_{21} = \frac{\partial g}{\partial w} = \frac{N}{\tau}$.

We transform equations (5), (6), and (7) to Laplace domain:

$$sW(s) = K_{11}W(s) + K_{12}W(s)e^{-s\tau} + K_{12}P(s)e^{-s\tau} \qquad (7)$$

$$sQ(s) = K_{21}W(s) \qquad (8)$$

$$sP(s) = aQ(s) - bP(s). \qquad (9)$$

From equation (7), (8), and (9), we can derive the characteristic equation of the system.

$$s^3 + a_1 s^2 + a_2 s + a_3 = 0 \qquad (10)$$

where

$$a_1 = -K_{11} - K_{12}e^{-s\tau} + b,\ a_2 = -K_{11}b, \text{ and } a_3 = -aK_{12}K_{21}e^{-s\tau}.$$

Since the real parts of a_1, a_2, and a_3 are positive, any roots of equation (10) must have a negative real part. Hence, the system is stable.

As an example, consider a network with capacity C = 100 Mbps = 12,500 packets/sec (for an average packet size of 1,000 bytes), and N = 1,000 flows. For T = 0.001 second, a = 0.0001 and b = 0.1, we obtain the Nyquist diagram of the transfer function of the system TCP/LQD shown Figure 1. Since this open loop diagram does not encircle the point (-1, 0), the closed-loop system TCP/LQD does not enclose the origin and does not have any poles (roots of equation (10)) in the right-half plane. Therefore, the system consisting of N TCP flows (plant) and an LQD (controller) is stable.

4 Experimental Methodology

To evaluate LQD we run experiments in the network testbed described in [16]. The network, illustrated in Figure 2, emulates a peering link between two Internet service provider (ISP) networks. In this network, we emulate a large population of users using a mix of TCP-based applications (described below).

The testbed consists of approximately 50 Intel processor based machines running FreeBSD 4.5. Machines execute synthetic traffic generation programs that produce synthetic TCP traffic based on measurements of TCP traffic on real network links [10]. The traffic is generated in such a way that it is statistically similar to the traffic on the measured link (*e.g.* the distributions of packet sizes, object sizes, active connections per second, throughput per second, *etc.* observed on the real network can be reproduced in the laboratory network [10, 16]). For this work we use two traffic generators: a synthetic HTTP generator [16] and a generator capable of reproducing the mix of application traffic seen on Abilene [10]. The HTTP workload is used to compare LQD results with previous studies. A novel aspect of this study is the consideration of general TCP traffic. For concreteness, to explain the basic experimental methodology, we focus here on the process of generating synthetic HTTP traffic. The generation of Abilene traffic is similar and is briefly described in Section 5.

End-systems in Figure 2 execute either a web request generator (a "browser") that emulates the browsing behavior of thousands of human users, or a web response generator (a "server") that responds to requests by transmitting an object back to the requesting machine. The browser and server machines have 100 Mbps Ethernet interfaces and are attached to switched VLANs with both 100 Mbps and 1 Gbps ports on 10/100/1000 Ethernet switches. The users and the servers they contact are evenly distributed across ISP1 and ISP2. At the core of this network are two router machines

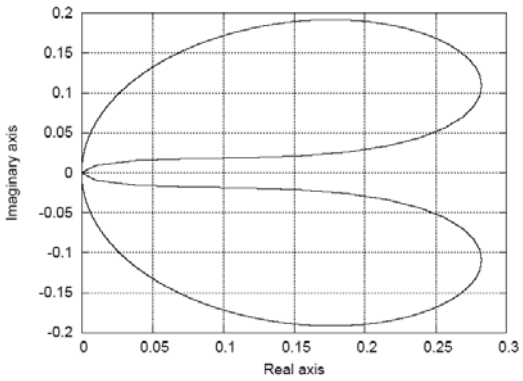

Figure 1: Nyquist diagram for the system TCP/LQD.

running the ALTQ extensions to FreeBSD. ALTQ extends IP-output queuing at the network interfaces to include alternative queue-management disciplines [12]. We used the ALTQ infrastructure to implement LQD, PI, and REM.

Each router has sufficient network interfaces to create either a point-to-point 100 Mbps Fast Ethernet network between the two routers or a point-to-point Gigabit Ethernet between the routers. The Gigabit Ethernet network is used as an uncongested network on which we perform calibration experiments to benchmark the traffic generators. To evaluate LQD and compare its performance to other AQM schemes, we create a congested 100 Mbps between the routers by changing static routes in the routers to use the Fast Ethernet interfaces rather than the gigabit interfaces.

So that we can emulate flows that traverse a longer network path than the one in our testbed, we use a locally-modified version of *dummynet* [22] to configure out-bound packet delays on browser machines. These delays emulate different round-trip times on *each* TCP connection (thus giving *per-flow* delays). Our version of *dummynet* delays all packets from each flow by the same randomly-chosen minimum delay as described in [16]. Thus while our network is fundamentally a "dumbbell" topology, our use of per-flow minimum round-trip times ensures a packet arrival process at the routers that mimics that found in wide-area networks (*e.g.*, is long-range dependent) [10, 16].

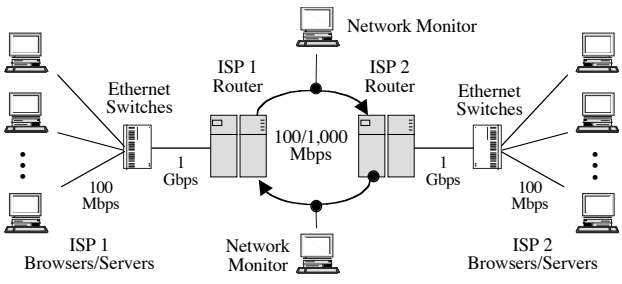

Figure 2: Experimental network setup.

4.1 Synthetic Generation of Web Traffic

The HTTP traffic we generate is based on an empirical model derived from a large-scale analysis of web traffic [23]. The model is an application-level description of how the HTTP 1.0 and 1.1 protocols are used. The model of web browsing is as described in [16, 23], however, here we note that the model is quite detailed. For example, the model captures the use of persistent HTTP connections as implemented in many contemporary browsers and servers, and distinguishes between web objects that are "top-level" (*e.g.*, HTML files) and objects that are embedded (*e.g.*, image files).

The model is expressed as a set of empirical distributions describing the elements necessary to generate synthetic HTTP workloads. For each request, a "browser" program generates a message of random size, sampled from the request size distribution, and sends it to an instance of a "server" program. The server returns a response whose size is a random sample from the distributions of response sizes for top-level or embedded requests. The empirical distribution of response sizes is heavy tailed such that while the median response size is approximately 10,000 bytes, responses as large as 10^9 bytes are also generated.

For each request/response pair, the browser program logs the response time. When all of the request/response pairs for a page have been completed, the browser enters a thinking state and makes no more requests for a random period of time sampled from a think time distribution.

4.2 Experimental Procedures

To evaluate LQD we performed experiments on the two ISP networks in our testbed connected with 100 Mbps links that we congest with varying degrees of traffic. To quantify the traffic load in each experiment we define *offered load* as the network traffic (in bits/second) resulting from emulating the browsing behavior of a fixed-size population of web users. More specifically, load is expressed as the long-term average throughput on an uncongested 1 Gbps link that would be generated by that user population. For example, to describe the load offered by emulating a population of 20,000 users evenly distributed on our network testbed, we would first emulate this user population on our network with the two ISP networks connected with a gigabit/second link and measure the average throughput in one direction on this link. The measured throughput, approximately 105 Mbps in this case, is our value of offered load.

Since experiments are ultimately performed with the two ISP networks connected at 100 Mbps, we ran a series of calibration experiments to determine how load on the gigabit network varied as a function of the number of emulated users. As expected, load varied linearly with number if emulated users (see [16] for details). Thus, for example, if we want to generate an offered load equal to 80% of the capacity of the 100 Mbps link (*i.e.*, 80 Mbps), the calibration experiments tell us that we need to emulate approximately 7,600 users in each ISP to consume 80% of the link in each direction. Note that as offered loads approach saturation of the 100 Mbps link, the actual link utilization will, in general, be less than the intended offered load. This is because as utilization increases, response times become longer and users have to wait longer before they can generate new requests and hence generate fewer requests per unit time.

Previous studies have shown that AQM is effective at high loads [16]. Therefore, each experiment was run using offered loads of 90%, 98%, or 105% of the capacity of the 100 Mbps link connecting the two router machines. It is important to emphasize again that terms like "105% load" are used as a shorthand notation for "a population of users that would generate a long-term average load of 105 Mbps on a 1 Gbps link." Thus our notion of offered load refers to the traffic generating *capacity* present in an experiment, not the actual load generated in an experiment. (The actual load generated in an experiment is a function of the performance of the AQM scheme used.) Each experiment was run for 120 minutes to ensure very large samples (over 10,000,000 request/response exchanges), but data were collected only during a 90-minute interval to eliminate startup effects and experiment termination synchronization anomalies.

4.3 Measures of Success

The primary metrics for comparing the performance of AQM schemes are loss-rate and a measure of the router's queue length. While we would like to measure and observe directly how routers' queues evolve over time when AQM is used, it is technically and semantically difficult to do so. The reason is that routers (both "real" routers and our PC routers) have multiple packet queues on the outbound path and only one of these queues is controlled by the AQM scheme. For example, line cards on routers and NICs on PCs buffer a potentially large number of outgoing packets. These buffers are different from the IP output queue managed by the router OS where AQM is applied [13]. For this reason, simple measures of the IP output queue can be misleading as more (or less) packets may also be queued at in the lower layer queue. Getting queue length data from the line card or NIC is difficult as it increases the workload of the processor on the card and hence can effect the card's performance (and thus bias experimental results). Moreover, measures of instantaneous queue length can be misleading as application performance depends on the sum of the queue lengths seen by each arriving packet from an application's connection. Measures of average queue length are also not good predictors of application performance unless the averaging is done on a per connection basis (one sample for each packet of a connection). The cost of gathering such data similarly effects router performance as this level of per-packet processing is not optimized and has high memory requirements.

To get around the difficulties of measuring and interpreting queue length, we adopt an indirect measure of queue length, namely end-to-end response times for a TCP application data unit exchange. Response time measures the combined effect of instantaneous queue length seen by a connection's packets and the loss-rate seen by a connection. For this reason it is an effective summary measure of AQM performance. We also report in a more summary manner, network-centric performance measures such as the fraction of IP datagrams dropped at the link queues, the link utilization on the bottleneck link, and the number of request/response exchanges completed in the experiment.

5. Experimental Results

We implemented LQD in the framework of ALTQ [12] and ran experiments in our laboratory network to evaluate it. We also implemented PI, REM, ARED, SFB, Blue, RIO-PS, AVQ, and AFD and compared LQD against them. Because of space considerations, we only show the results for LQD versus PI and REM. (In all experiments, the performance of PI and REM always dominated that of the other algorithms. Thus the most significant comparisons to be made for LQD are against PI and REM.) We also include results from experiments with drop-tail FIFO queue management to illustrate the performance of no AQM, and results from experiments on the uncongested gigabit network to illustrate the best possible performance.

For PI, REM, and LQD, we performed extensive initial experiments to determine "optimal" target queue lengths. Based on these experiments, two target queue lengths were chosen: 24 and 240 packets. These were chosen to provide two operating points; one that potentially yields low latency (24) and one that potentially provides high link utilization (240). We found that PI performs best at a target queue length of 240 at all loads. On the other hand, REM performs best at a target queue length of 24. We only report results for PI and REM with those target queue lengths. LQD obtains its best performance with a target queue length of 24 overall although it obtains almost comparable performance with a target queue length of 240. When comparing LQD against PI and REM, we used a target queue length of 24 for LQD and REM, and a target queue length of 240 for PI. For PI and REM, we also used the parameter settings that were recommended by their inventors. We also experimented with different parameter settings for each AQM scheme but only reported the results for the best settings here due to space limitations. In all cases we set the maximum queue size to a high number of packets that ensured tail drops did not occur. (Recall that the target queue length does not represent the amount of buffering present in the router.)

5.1 Experimental Results with Web Traffic

Figures 3-8 give the results for LQD, PI, REM, and drop-tail FIFO. They show the cumulative distribution functions (CDFs) and complementary cumulative distribution functions (CCDFs) for response times at offered loads of 90%, 98%, and 105% respectively. We also report other statistics for the experiments in Table 1.

At 90% load, REM obtained approximately the same performance as drop-tail for the shortest 80% of responses and gave worse performance than drop-tail for the remaining 20% of responses. On the other hand, PI gave worse performance than drop-tail for the shortest 80% of responses but achieved better performance than drop-tail for the remaining 20% of responses. These results demonstrate the tradeoff between minimizing queuing delay and loss rate for improving application performance. While REM with a target queue length of 24 managed to maintain a short queue, it also inflicted a higher loss rate on connections (see Table 1). These are the connections that experienced worse performance than under drop-tail. On the other hand, PI with a target queue threshold of 240 reduced packet loss rate but also increased queuing delay and response times for applications' request-response exchanges.

LQD managed to balance queuing delay and loss rate and simultaneously gave good performance for response times, loss-rate and link utilization. LQD absorbs transient congestion (and avoids dropping packets aggressively) by allowing routers' queue to grow temporarily. However, when persistent congestion occurs, LQD still maintains stabilized routers' queue by increasing the drop probability appropriately. Figure 4 shows the tail of the response time distribution and shows that response times were best under LQD for all request-response exchanges except for a handful of large responses having response times larger than 1,000 seconds.

At loads of 98% and 105%, there is clear performance superiority for LQD over other AQM designs. We also note that all AQM designs provide performance superior or comparable to drop-tail at these loads. This result demonstrates the benefits of AQM. We also see in Table 1 that loss rate increases for all AQM designs as load increases. However, LQD obtained the lowest loss rate among all AQM designs. Link utilization was also highest under LQD at these loads.

5.2 Experiments with General TCP Traffic

While results for LQD from previous experiments are encouraging, they are limited to only Web traffic. To demonstrate the generality of our results, we repeated our experiment using synthetic traffic that is derived from the full mix of TCP connections captured on Internet links. We use a 2-hour packet trace taken on an Abilene (Internet 2) link between Cleveland and Indianapolis. The data to drive this experiment was acquired from the NLANR repository. The packet trace is filtered for *all* TCP connections including HTTP, FTP, SMTP, NNTP, and peer-to-peer file-sharing traffic. The synthetic traffic generated in our network represents the characteristics of existing Internet backbone traffic as seen by routers in real network and provides the most

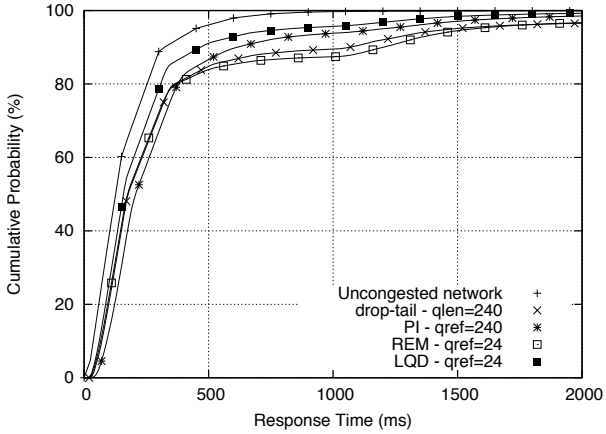

Figure 3: Comparison of the response time distributions of all schemes at 90% load.

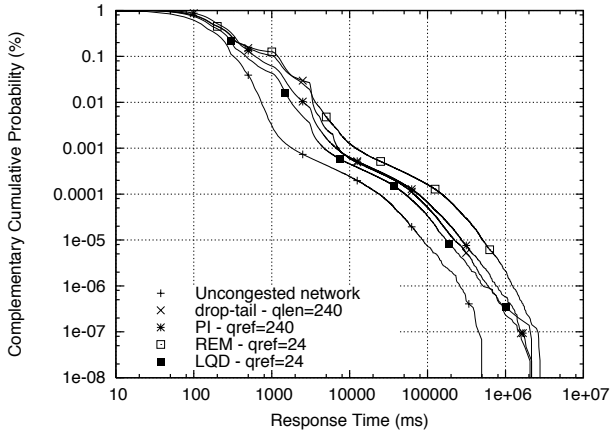

Figure 4: Tail of response time distributions for all schemes at 90% load.

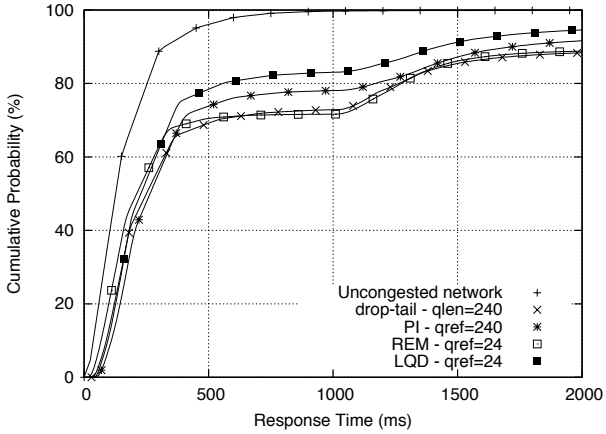

Figure 5: Comparison of the response time distributions of all schemes at 98% load.

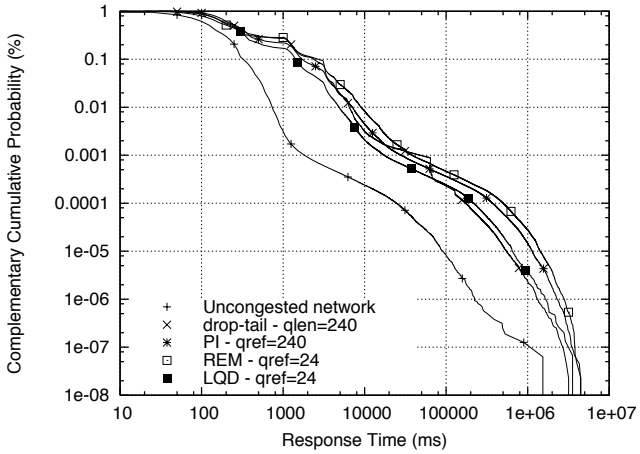

Figure 6: Tail of response time distributions for all schemes at 98% load.

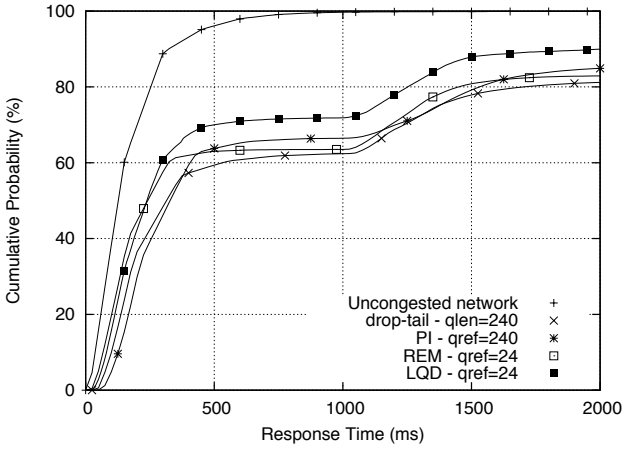

Figure 7: Comparison of the response time distributions of all schemes at 105% load.

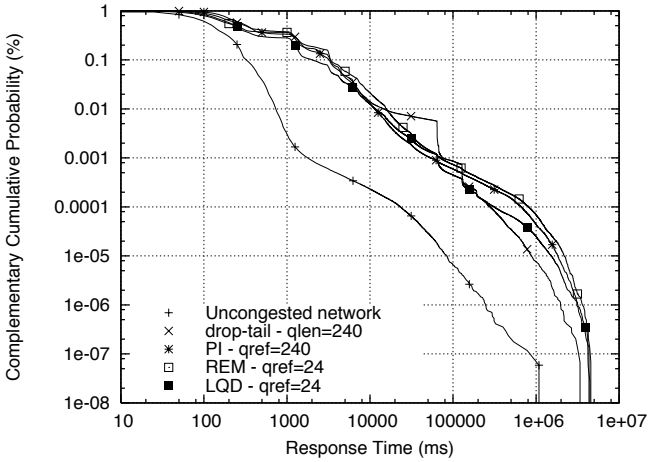

Figure 8: Tail of response time distributions for all schemes at 105% load.

realistic method for evaluating AQM designs in a laboratory network. The application used to generate synthetic TCP traffic using packet traces is called *tmix* and is described in [10]. We only provide a high-level description here.

Table 1: Loss, completed requests, and link utilizations.

	Offered Load	Loss ratio (%)	Completed requests (millions)	Link utilization/ throughput (Mbps)
Uncongested 1 Gbps network (drop-tail)	90%	0	15.0	91.3
	98%	0	16.2	98.2
	105%	0	17.3	105.9
drop-tail queue size = 240	90%	1.8	14.6	89.9
	98%	6.0	15.1	92.0
	105%	8.8	15.0	92.4
PI $q_{ref} = 24$	90%	1.3	14.4	87.9
	98%	3.9	15.1	89.3
	105%	6.5	15.1	89.9
PI $q_{ref} = 240$	90%	0.1	14.7	87.2
	98%	3.7	14.9	90.0
	105%	6.9	15.0	90.5
LQD $q_{ref} = 24$	90%	0.4	14.7	88.5
	98%	2.7	15.3	91.6
	105%	4.9	15.6	91.9
LQD $q_{ref} = 240$	90%	0.2	14.7	88.3
	98%	2.6	15.3	91.9
	105%	5.1	15.7	92.1
REM $q_{ref} = 24$	90%	1.8	14.4	86.4
	98%	5.0	14.5	87.6
	105%	7.7	14.6	87.5
REM $q_{ref} = 240$	90%	3.3	14.0	83.3
	98%	5.4	14.4	86.2
	105%	7.3	14.6	87.7

tmix models traffic sources as network-independent entities. The model is based on two endpoints exchanging application data units (ADUs) defined by their specific application-level protocol. The structure and sequencing of ADUs is extracted from packet header traces via a "reverse compilation" process. ADUs are aggregated into connections with ADU exchanges within a connection separated by measured think times. These connections, in essence a specification of how an application used TCP at the socket layer to transmit data, are replayed by a traffic generator. The connection descriptions form a source-level model of how TCP is used by the applications found on a given network.

The connection descriptions are replayed into the network to generate synthetic traffic. During the replay, each TCP connection is reproduced as a sequence of data unit exchanges and think times, beginning at the same instant and the same order as it appears in the original trace. We define response time in this experiment as the time interval necessary to complete the exchange of data units between two endpoints. The degree of congestion induced in a network via *tmix* is a function of the load on the original traced network and the capacity of the replay network. The congestion can be controlled via a *scaling* process that dilates (expands) or compresses the range of TCP connection start times in the trace. Here we show experiments with two different scalings: a "2.0" scaling (start times expanded by a factor of 2 thus giving a less congested load than the original network), and a "1.75" scaling (giving a more congested load than the 2.0 scaling). The choice of scaling parameter followed a calibration process similar to that described in Section 3 for HTTP. The actual offered loads induced in the network for 2.0 scaling were approximately 105.3 Mbps on average in one direction and 91.2 Mbps on the reverse path. (Note that an interesting aspect of Abilene traffic for experiments is that it is not symmetrical between forward and reverse paths.) For 1.75 scaling, the offered loads were approximately 119 Mbps (forward path) and 104 Mbps (reverse path).

Figures 9-12 show the response time results of AQM performance on Abilene traffic. For comparison purposes, we also report results for the uncongested network and for drop-tail FIFO at a queue depth of 240. We see the benefits of AQM designs over drop-tail (except for PI which obtained slightly worse performance than drop-tail for the shortest 75% of responses at both offered loads). LQD again obtained the best performance among all schemes and came closest to the performance obtained on an uncongested network. The summary statistics for these experiments are included in Table 2. The efficiency of LQD is also reflected in its lowest loss rate and highest link utilization among all AQM designs.

6. Summary and Conclusions

Active queue management (AQM) in routers has been proposed as a solution to some of the scalability issues associated with TCP's pure end-to-end approach to congestion control. Our recent study of AQM schemes [16] demonstrated that the AQM algorithms are effective in reducing the response times of web request/response exchanges as well as increasing link throughput and reducing loss rates. However, ECN was required for these results. Since ECN is currently not widely deployed in the Internet, we argue in this paper that AQM schemes should be cautious when they convey congestion signal to end systems by dropping packets. While we appreciate the benefits of having stabilized router queues, we argue that controlling queue should not be the ultimate goal for AQM designs. Furthermore, this goal should not be achieved by dropping packets aggressively.

We presented an alternate approach, called *loss and queuing delay* (LQD) controller, that enables a more flexible framework in managing routers' resources. LQD allows to balance queuing delay and loss rate at a router to improve network and application performance. Within this framework, LQD can obtain a low loss rate by allowing routers' queue to grow temporarily when transient congestion occurs. When congestion is persistent, LQD can still control router queues by increasing the packet loss rate appropriately.

We evaluated LQD and compared it to prominent AQM designs such as PI and REM under realistic conditions. Our

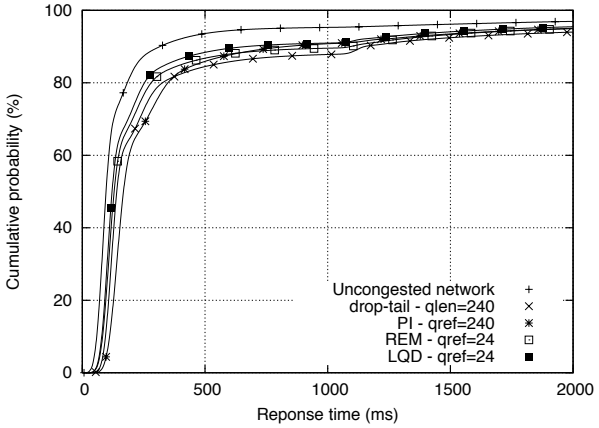

Figure 9: Comparison of response time distributions (body) of all schemes under *tmix* traffic with 2.0 scaling.

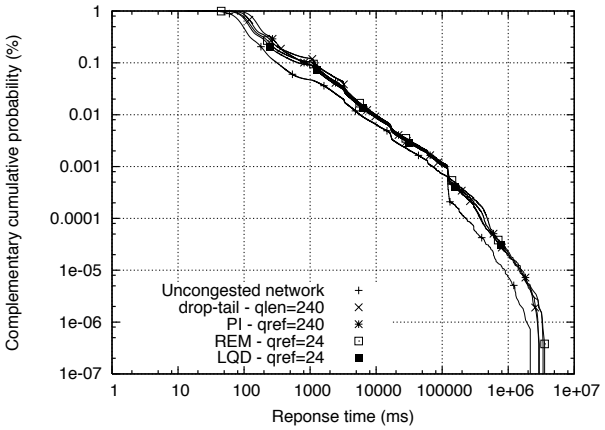

Figure 10: Comparison of response time distributions (tail) of all schemes under *tmix* traffic with 2.0 scaling.

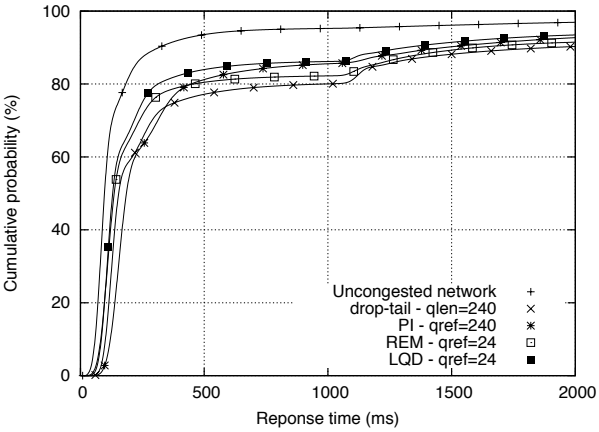

Figure 11: Comparison of response time distributions (body) of all schemes under *tmix* traffic with 1.75 scaling.

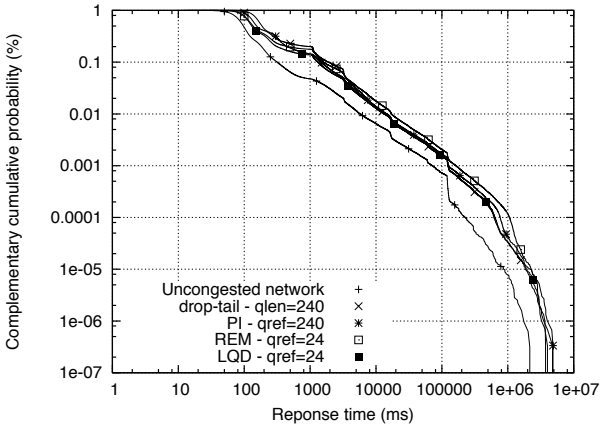

Figure 12: Comparison of response time distributions (tail) of all schemes under *tmix* traffic with 1.75 scaling.

experiments showed that both network and applications benefit from our approach and LQD outperforms the other AQM designs. Response times for request/response exchanges were most improved under LQD. Further, LQD obtained higher link utilization and lower loss rate than existing AQM schemes. We believe that our results open a new direction and inspire a new focus in AQM design.

7. Acknowledgements

We would like to thank the anonymous referees as well as Lisa Fowler for their constructive comments and suggestions for revising this paper.

This work was supported in parts by the National Science Foundation (grants CCR-0208924, EIA-0303590, and ANI-0323648), Cisco Systems Inc., and the IBM Corporation.

8. References

[1] S. Athuraliya, V. H. Li, S.H. Low, Q. Yin, *REM: Active Queue Management*, IEEE Network, Vol. 15, No. 3, May 2001, pp. 48-53.

[2] B. Braden, *et al.*, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[3] A. Feldmann, A. Gilbert, and W. Willinger, *Data networks as cascades: Explaining the multifractal nature of Internet WAN traffic*, ACM SIGCOMM 1998.

[4] W. Feng, D. Kandlur, D. Saha, and K. Shin, *Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness*, Proc., IEEE INFOCOM 2001, April 2001.

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin, *Blue: An Alternative Approach To Active Queue Management*, in Proc. of NOSSDAV 2001, June 2001.

[6] S. Floyd, *Congestion Control Principles*, RFC 2914, September 2000.

[7] S. Floyd, R. Gummadi, S. Shenker, *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*, *http://www.icir.org/floyd/papers/adaptiveRed.pdf*, August 1, 2001.

[8] S. Floyd, and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance,* IEEE/ACM Transactions on Networking, Vol. 1 No. 4, August 1993, pp. 397-413.

[9] L. Guo and I. Matta: *The War between Mice and Elephants*, Proc., ICNP 2001, Nov. 2001, pp. 180-188.

Table 2: Summary statistics for experiments using *tmix*.

	Rate scaling	Completed exchanges (millions)	Loss rate (forward path) (%)	Loss rate (reverse path) (%)	Link throughput (forward path) (Mbps)	Link throughput (reverse path) (Mbps)
Uncongested	2.0	2.8	0.0	0.0	105.3	91.2
	1.75	3.2	0.0	0.0	119.1	104.5
drop-tail qlen = 240	2.0	2.6	3.7	0.9	90.8	85.7
	1.75	3.0	7.2	2.7	91.0	89.1
PI w/ q_{ref} = 240	2.0	2.6	1.7	0.6	87.9	83.6
	1.75	3.0	4.2	2.4	86.5	84.9
LQD q_{ref} = 24	2.0	2.7	1.4	0.6	90.6	86.7
	1.75	3.0	4.0	2.1	90.5	89.5
REM w/ q_{ref} = 24	2.0	2.6	2.1	0.8	84.3	81.4
	1.75	3.0	5.4	3.4	83.4	81.6

[10] F. Hernandez Campos, K. Jeffay, and F.D. Smith, *Generating Realistic TCP Workloads*, Proc., Computer Measurement Group Intl. Conf., Las Vegas, NV, Dec. 2004, pp. 273-284.

[11] C.V. Hollot, Vishal Misra, Don Towsley, and W. Gong, *On Designing Improved Controllers for AQM Routers Supporting TCP Flows*, Proc., IEEE Infocom 2001, pp. 1726-1734.

[12] C. Kenjiro, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers*, USENIX 1998 Annual Technical Conf., June 1998, pp. 247-258.

[13] C. Kenjiro, *Fitting theory into reality in the ALTQ case*, ASIA BSD conference, Taipei, Taiwan, March 2004.

[14] S. Kunniyur and R. Srikant, *Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management*, Proc., ACM SIGCOMM 2001, pp. 123-134.

[15] L. Le, *Understanding the Effects of Active Queue Management on Web and General TCP Applications*, Ph.D. thesis, University of North Carolina, 2005.

[16] L. Le, J. Aikat, K. Jeffay, F. D. Smith, *The Effects of Active Queue Management on Web Performance*, Proc., ACM SIGCOMM 2003, Aug. 2003, pp. 265-276.

[17] S. Low, F. Paganini, J. Wang, S. Adlakha, J. Doyle, *Dynamics of TCP/RED and a Scalable Control*, IEEE Infocom 2002.

[18] V. Misra, W. Gong, and D Towsley, *A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED,* ACM SIGCOMM 2000.

[19] A. Medina, M. Allman, and S. Floyd, *Measuring Interactions Between Transport Protocols and Middleboxes*, ACM Internet Measurement Conference 2004, August 2004.

[20] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, *Approximate Fairness through Differential Dropping*, ACM CCR, April 2003, pp. 23-39.

[21] V. Paxson, and S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, Vol. 3 No. 3, pp. 226-244, June 1995.

[22] L. Rizzo, *Dummynet: A simple approach to the evaluation of network protocols*, ACM CCR, Vol. 27, No. 1, January 1997.

[23] F.D. Smith, F. Hernandez Campos, K. Jeffay, and D. Ott, *What TCP/IP Protocols Headers Can Tell Us About The Web*, Proc., ACM SIGMETRICS 2001, June 2001, pp. 245-256.