The Effects of Active Queue Management on Web Performance

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

ABSTRACT

We present an empirical study of the effects of active queue management (AQM) on the distribution of response times experienced by a population of web users. Three prominent AQM schemes are considered: the Proportional Integrator (PI) controller, the Random Exponential Marking (REM) controller, and Adaptive Random Early Detection (ARED). The effects of these AQM schemes were studied alone and in combination with Explicit Congestion Notification (ECN). Our major results are:

1. For offered loads up to 80% of bottleneck link capacity, no AQM scheme provides better response times than simple drop-tail FIFO queue management.
2. For loads of 90% of link capacity or greater when ECN is not used, PI results in a modest improvement over drop-tail and the other AQM schemes.
3. With ECN, both PI and REM provide significant response time improvement at offered loads above 90% of link capacity. Moreover, at a load of 90% PI and REM with ECN provide response times competitive to that achieved on an unloaded network.
4. ARED with recommended parameter settings consistently resulted in the poorest response times which was unimproved by the addition of ECN.

We conclude that without ECN there is little end-user performance gain to be realized by employing the AQM designs studied here. However, with ECN, response times can be significantly improved. In addition it appears likely that provider links may be operated at near saturation levels without significant degradation in user-perceived performance.

Categories and Subject Descriptors

C.2.2 **[Computer Systems Organization]**: Computer Communication Networks — *Network Protocols*.

General Terms

Algorithms, Measurement, Performance, Experimentation.

Keywords

Congestion control, Active queue management, Web performance.

1 INTRODUCTION AND MOTIVATION

The random early detection (RED) algorithm, first described ten years ago [7], inspired a new focus for congestion control research on the area of *active queue management* (AQM). The common goal of all AQM designs is to keep the average queue size small in routers. This has a number of desirable effects including (1) providing queue space to absorb bursts of packet arrivals, (2) avoiding lock-out and bias effects from a few flows dominating queue space, and (3) providing lower delays for interactive applications such as web browsing [3].

All AQM designs function by detecting impending queue buildup and notifying sources before the queue in a router overflows. The various designs proposed for AQM differ in the mechanisms used to detect congestion and in the type of control mechanisms used to achieve a stable operating point for the queue size. Another dimension that has a significant impact on performance is how the congestion signal is delivered to the sender. In today's Internet where the dominant transport protocol is TCP (which reacts to segment loss as an indicator of congestion), the signal is usually delivered implicitly by dropping packets at the router when the AQM algorithm detects queue buildup. An IETF proposed standard adds an explicit signalling mechanism, called *explicit congestion notification* (ECN) [12], by allocating bits in the IP and TCP headers for this purpose. With ECN a router can signal congestion to an end-system by "marking" a packet (setting a bit in the header).

In this work we report the results of an empirical evaluation of three prominent examples of AQM designs. These are the Proportional Integrator (PI) controller [8], the Random Exponential Marking (REM) controller [2] and a contemporary redesign of the classic RED controller, Adaptive RED [6] (here called ARED). While these designs differ in many respects, each is an attempt to realize a control mechanism that achieves a stable operating point for the size of the router queue. Thus a user of each of these mechanisms can determine a desired operating point for the control mechanism by simply specifying a desired target mean queue size. Choosing the desired queue size may represent a tradeoff between link utilization and queuing delay — a short queue reduces latency at the router but setting the target queue size too small may reduce link utilization by causing the queue to drain needlessly.

Our goal in this study was first and foremost to compare the performance of control theoretic AQM algorithms (PI and REM) with the more traditional randomized dropping found in RED. For performance metrics we chose both user-centric measures of performance such as response times for the request-response transactions that comprise Web browsing, as well as more traditional metrics such as achievable link utilization and loss rates. The distribution of response times that would be experienced by a population of web users is used to assess the user-perceived performance of the AQM schemes. Link utilization is used to assess the impact on network resources. Of particular interest was the implication of ECN support on performance. ECN requires the participation of end-systems in the AQM scheme and hence it is important to quantify the performance gain to be had at the expense of a more complex protocol stack and migration issues for the end-system.

SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.

Our experimental platform was a laboratory testbed consisting of a large collection of computers arranged to emulate a peering point between two ISPs operated at 100 Mbps (see Figure 1). We emulated the Web browsing behavior of tens of thousands of users whose traffic transits the link connecting the ISPs and investigated the performance of each AQM scheme in the border-routers connecting the ISPs. Each scheme was investigated both with and without ECN support across a variety of AQM parameter settings that represented a range of target router-queue lengths. For each target queue length we varied the offered load on the physical link connecting the ISPs to determine how (or if) AQM performance was affected by load.

Our results were that for offered loads up to 80% of the bottleneck link capacity, no AQM scheme provided better response time performance than simple drop-tail FIFO queue management. In addition, all schemes resulted in similar loss rates and link utilization. For offered loads above 80% of link capacity there was an advantage to employing control theoretic AQM. When ECN is not used, at offered loads of 90% of link capacity, PI resulted in a modest improvement over drop-tail and the other AQM schemes. Web browsing response time was improved for responses requiring more than approximately 400 milliseconds to complete but at the cost of slightly lower achievable link utilization (compared to drop-tail). Of particular note was the fact that without ECN, PI gave performance superior to REM.

Our most striking result is that with ECN, both REM and PI significantly outperform drop-tail at 90% load and provide response time performance that is competitive to that achieved on an unloaded network. The improved response time performance, however, comes at some loss of achievable link utilization. In light of these results, an additional striking result was the fact that the addition of ECN did not improve ARED performance. ARED consistently resulted in the poorest response time performance across all offered loads and resulted in the lowest link utilizations.

We conclude that without ECN there is little end-user or provider performance gain to be realized by employing the AQM algorithms studied here. However, with ECN performance can be significantly improved. In addition, our experiments provide evidence that provider links may be operated at near saturation levels (90% average utilization with bursty traffic sources) without significant degradation in user-perceived performance and with only very modest decreases in link utilization (when compared to drop-tail). Thus unlike a similar earlier study [4] which was negative on the use of AQM, we view the ECN results as a significant indicator that the stated goals of AQM can be realized in practice.

While the results of this study are intriguing, the study was nonetheless limited. The design space of AQM schemes is large with each algorithm typically characterized by a number of independent parameters. We limited our consideration of AQM algorithms to a comparison between two classes of algorithms: those based on control theoretic principles and those based on the original randomized dropping paradigm of RED. Moreover, we studied a link carrying only Web-like traffic. More realistic mixes of HTTP and other TCP traffic as well as traffic from UDP-based applications need to be examined.

The following section reviews the salient design principles of current AQM schemes and reviews the major algorithms that have been proposed. Section 3 presents our experimental methodology and discusses the generation of synthetic Web traffic. Section 4 presents our results for AQM with packet drops and Section 5 presents our results for AQM with ECN. The results are discussed in Section 6. We conclude in Section 7 with a summary of our major results.

2 BACKGROUND AND RELATED WORK

The original RED design uses a weighted-average queue size as a measure of congestion. 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 marked or dropped. (The actual size of the queue must be greater than max_{th} to absorb transient bursts of packet arrivals.) A modification to the original design introduced a "gentle mode" in which the mark or drop probability increases linearly between max_p and 1 as the average queue length varies between max_{th} and 2 x max_{th}. This fixes a problem in the original RED design caused by the non-linearity in drop probability (increasing from max_p to 1.0 immediately when max_{th} is reached).

A weakness of RED is that it does not take into consideration the number of flows sharing a bottleneck link [5]. Given the TCP congestion control mechanism, a packet mark or drop reduces the offered load by a factor of $(1 - 0.5n^{-1})$ where n is the number of flows sharing the bottleneck link. Thus, RED is not effective in controlling the queue length when n is large. On the other hand, RED can be too aggressive and can cause under-utilization of the link when n is small. Feng *et al*. concluded that RED needs to be tuned for the dynamic characteristics of the aggregate traffic on a given link [5]. They proposed a self-configuring algorithm for RED by adjusting max_p every time the average queue length falls out of the target range between min_{th} and max_{th}. When the average queue length is smaller than min_{th}, max_p is decreased multiplicatively to reduce RED's aggressiveness in marking or dropping packets; when the queue length is larger than max_{th}, max_p is increased multiplicatively. Floyd *et al.* improved upon this original adaptive RED proposal by replacing the MIMD (multiplicative increase multiplicative decrease) approach with an AIMD (additive increase multiplicative decrease) approach [6]. They also provided guidelines for choosing min_{th}, max_{th}, and the weight for computing a target average queue length. The RED version that we implemented and studied in our work (referred to herein as "ARED") includes both the adaptive and gentle refinements to the original design. It is based on the description given in [6].

In [11], Misra *et al.* applied control theory to develop a model for TCP and AQM dynamics and used this model to analyze RED. They pointed out two limitations in the original RED design: (1) RED is either unstable or has slow responses to changes in network traffic, and (2) RED's use of a weighted-average queue length to detect congestion and its use of loss probability as a feedback signal to the senders were flawed. Because of this, in overload situations, flows can suffer both high delay and a high packet loss rate. Hollot *et al.* simplified the TCP/AQM model to a linear system and designed a Proportional Integrator (PI) controller that regulates the queue length to a target value called the "queue reference," q_{ref} [8]. The PI controller uses instantaneous samples of the queue length taken at a constant sampling frequency as its input. The drop probability is computed as

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

where $p(kT)$ is the drop probability at the k^{th} sampling interval, $q(kT)$ is the instantaneous sample of the queue length and T is $1/$*sampling-frequency*. A close examination of this equation shows that the drop probability increases in sampling intervals when the queue length is higher than its target value. Furthermore, the drop probability also increases if the queue has grown since the last sample (reflecting an increase in network traffic). Conversely, the drop probability in a PI controller 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 equation depend on the link capacity, the maximum RTT and the expected number of active flows using the link.

In [2], Athuraliya *et al*. proposed the Random Exponential Marking (REM) AQM scheme. REM periodically updates a congestion measure called "price" that reflects any mismatch between packet arrival and departure rates at the link (*i.e*., the difference between the demand and the service rate) and any queue size mismatch (*i.e*., the difference between the actual queue length and its target value). The measure (p) is computed by:

$$p(t) = \max(0, p(t-1) + \gamma \times (\alpha \times (q(t) - q_{ref})) + x(t) - c))$$

where c is the link capacity (in packet departures per unit time), $p(t)$ is the congestion measure, $q(t)$ is the queue length, and $x(t)$ is the packet arrival rate, all determined at time t. As with ARED and PI, the control target is only expressed by the queue size.

The mark/drop probability in REM is defined as $prob(t) = 1 - \phi^{-p(t)}$, where $\phi > 1$ is a constant. In overload situations, the congestion price increases due to the rate mismatch and the queue mismatch. Thus, 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. It is easy to see that a positive rate mismatch over a time interval will cause the queue size to increase. Conversely, a negative rate mismatch over a time interval will drain the queue. Thus, REM is similar to PI because the rate mismatch can be detected by comparing the instantaneous queue length with its previous sampled value. Furthermore, when drop or mark probability is small, the exponential function can be approximated by a linear function [1].

3 EXPERIMENTAL METHODOLOGY

For our experiments we constructed a laboratory network that emulates the interconnection between two Internet service provider (ISP) networks. Specifically, we emulate one peering link that carries web traffic between sources and destinations on both sides of the peering link and where the traffic carried between the two ISP networks is evenly balanced in both directions.

The laboratory network used to emulate this configuration is shown in Figure 1. All systems shown in this figure are Intel-based machines running FreeBSD 4.5. At each edge of this network are a set of 14 machines that run instances of a Web request generator (described below) each of which emulates the browsing behavior of thousands of human users. Also at each edge of the network is another set of 8 machines that run instances of a Web response generator (also described below) that creates the traffic flowing in response to the browsing requests. A total of 44 traffic generating machines are in the testbed. In the remainder of this paper we refer to the machines running the Web request generator simply as the "browser machines" (or "browsers") and the machines running the Web response generator as the "server machines" (or "servers"). 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 3Com 10/100/1000 Ethernet switches.

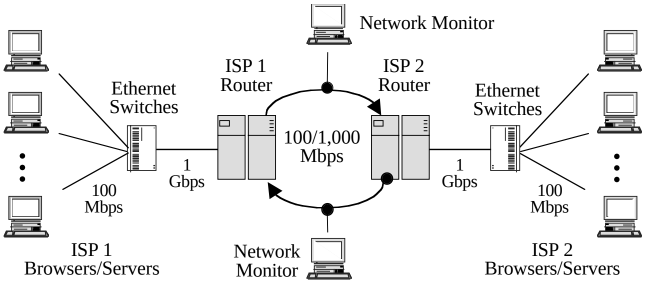

Figure 1: Experimental network setup.

At the core of this network are two router machines running the ALTQ extensions to FreeBSD. ALTQ extends IP-output queuing at the network interfaces to include alternative queue-management disciplines [10]. We used the ALTQ infrastructure to implement PI, REM, and ARED. The routers are 1 GHz Pentium IIIs with over 1 GB of memory. Each router has one 1000-SX fiber Gigabit Ethernet NIC attached to one of the 3Com switches. Each router also has three additional Ethernet interfaces (a second 1000-SX fiber Gigabit Ethernet NIC and two 100 Mpbs Fast Ethernet NICs) configured to create point-to-point Ethernet segments that connect the routers as shown in Figure 1. When conducting measurements to calibrate the traffic generators on an un-congested network, static routes are configured on the routers so that all traffic uses the full-duplex Gigabit Ethernet segment. When we need to create a bottleneck between the two routers, the static routes are reconfigured so that all traffic flowing in one direction uses one 100 Mbps Ethernet segment and all traffic flowing in the opposite direction uses the other 100 Mbps Ethernet segment.[1] These configurations allow us to emulate the full-duplex behavior of the typical wide-area network link.

Another important factor in emulating this network is the effect of end-to-end latency. We use a locally-modified version of the *dummynet* [9] component of FreeBSD to configure out-bound packet delays on browser machines to emulate different round-trip times on *each* TCP connection (giving *per-flow* delays). This is accomplished by extending the *dummynet* mechanisms for regulating per-flow bandwidth to include a mode for adding a randomly-chosen minimum delay to all packets from each flow. The same minimum delay is applied to all packets in a given flow (identified by IP addressing 5-tuple). The minimum delay in milliseconds assigned to each flow is sampled from a discrete uniform distribution on the range [10, 150] (a mean of 80 milliseconds). The minimum and maximum values for this distribution were chosen to approximate a typical range of Internet round-trip times within the continental U.S. and the uniform distribution ensures a large variance in the values selected over this range. We configured the *dummynet* delays only on the browser's outbound packets to simplify the experimental setup. Most of the data transmitted in these experiments flow from the server to the browser and the TCP con-

[1] We use two 100 Mbps Ethernet segments and static routes to separate the forward and reverse path flows in this configuration so we can use Ethernet hubs to monitor the traffic in each direction independently. Traffic on the Gigabit link is monitored using passive fiber splitters to monitor each direction independently.

gestion control loop at the server (the one AQM causes to react) is influenced by the total RTT, not by asymmetry in the delays relative to the receiver's side. Because these delays at the browsers effectively delay the ACKs received by the servers, the round-trip times experienced by the TCP senders (servers) will be the combination of the flow's minimum delay and any additional delay introduced by queues at the routers or on the end systems. (End systems are configured to ensure no resource constraints were present hence delays there are minimal, ~1 millisecond.) A TCP window size of 16K bytes was used on all the end systems because widely used OS platforms, *e.g.*, most versions of Windows, typically have default windows this small or smaller.

The instrumentation used to collect network data during experiments consists of two monitoring programs. One program monitors the router interface where we are examining the effects of the AQM algorithms. It creates a log of the queue size (number of packets in the queue) sampled every 10 milliseconds along with complete counts of the number of packets entering the queue and the number dropped. Also, a link-monitoring machine is connected to the links between the routers (through hubs on the 100 Mbps segments or fiber splitters on the Gigabit link). It collects (using a locally-modified version of the *tcpdump* utility) the TCP/IP headers in each frame traversing the links and processes these in real-time to produce a log of link utilization over selected time intervals (typically 100 milliseconds).

3.1 Web-Like Traffic Generation

The traffic that drives our experiments is based on a recent large-scale analysis of web traffic [13]. The resulting model is an application-level description of the critical elements that characterize how HTTP/1.0 and HTTP/1.1 protocols are used. It is based on empirical data and is intended for use in generating synthetic Web workloads. An important property of the model is that it reflects the use of persistent HTTP connections as implemented in many contemporary browsers and servers. Further, the analysis presented in [13] distinguishes between Web objects that are "top-level" (typically an HTML file) and those that are embedded objects (*e.g.*, an image file). At the time these data were gathered, approximately 15% of all TCP connections carrying HTTP protocols were effectively persistent (were used to request two or more objects) but more than 50% of all objects (40% of bytes) were transferred over these persistent connections.

The model is expressed as empirical distributions describing the elements necessary to generate synthetic HTTP workloads. The elements of the model that have the most pronounced effects on generated traffic are summarized in Table 1. Most of the behavioral elements of Web browsing are emulated in the client-side request-generating program (the "browser"). Its primary parameter is the number of emulated browsing users (typically several hundred to a few thousand). For each user to be emulated, the program implements a simple state machine that represents the user's state as either "thinking" or requesting a web page. If requesting a web page, a request is made to the server-side portion of the program (executing on a remote machine) for the primary page. Then requests for each embedded reference are sent to some number of servers (the number of servers and number of embedded references are drawn as random samples from the appropriate distributions). The browser also determines the appropriate usage of persistent and non-persistent connections; 15% of all new connections are randomly selected to be persistent. Another random selection from the distribution of requests per persistent connection is used to determine how many requests will use each persistent connection. One other parameter of the program is the number of parallel TCP connections allowed on behalf of each browsing user to make embedded requests within a page. This parameter is used to mimic the parallel connections used in Netscape (typically 4) and Internet Explorer (typically 2).

Table 1: Elements of the HTTP traffic model.

Element	Description
Request size	HTTP request length in bytes
Response size	HTTP reply length in bytes (top-level & embedded)
Page size	Number of embedded (file) references per page
Think time	Time between retrieval of two successive pages
Persistent connection use	Number of requests per persistent connection
Servers per page	Number of unique servers used for all objects in a page
Consecutive page retrievals	Number of consecutive top-level pages requested from a given server

For each request, a message of random size (sampled from the request size distribution) is sent over the network to an instance of the server program. This message specifies the number of bytes the server is to return as a response (a random sample from the distribution of response sizes depending on whether it is a top-level or embedded request). The server sends this number of bytes back through the network to the browser. The browser is responsible for closing the connection after the selected number of requests have completed (1 request for non-persistent connections and a random variable greater than 1 for persistent connections). For the experiments reported here, the server's "service time" is set to zero so the response begins as soon as the request message has been received and parsed. This very roughly models the behavior of a Web server or proxy having a large main-memory cache with a hit-ratio near 1. For each request/response pair, the browser program logs its response time. Response time is defined as the elapsed time between either the time of the socket *connect()* operation (for a non-persistent connection) or the initial request (on a persistent connection) or the socket *write()* operation (for subsequent requests on a persistent connection) and the time the last byte of the response is returned. Note that this response time is for each element of a page, not the total time to load all elements of a page.

When all the request/response pairs for a page have been completed, the emulated browsing user enters the thinking state and makes no more requests for a random period of time sampled from the think-time distribution. The number of page requests the user makes in succession to a given server machine is sampled from the distribution of consecutive page requests. When that number of page requests has been completed, the next server to handle the next top-level request is selected randomly and uniformly from the set of active servers. The number of emulated users is constant throughout the execution of each experiment.

3.2 Experiment Calibrations

Offered load for our experiments is defined as the network traffic resulting from emulating the browsing behavior of a fixed-size population of web users. It is expressed as the long-term average throughput (bits/second) on an un-congested link that would be generated by that user population. There are three critical elements of our experimental procedures that had to be calibrated before performing experiments:

1. Ensuring that no element on the end-to-end path represented a primary bottleneck other than the links connecting the two routers when they are limited to 100 Mbps,
2. The offered load on the network can be predictably controlled using the number of emulated users as a parameter to the traffic generators, and
3. Ensuring that the resulting packet arrival time-series (*e.g.*, packet counts per millisecond) is long-range dependent as expected because the distribution of response sizes is a heavy-tailed distribution [13].

To perform these calibrations, we first configured the network connecting the routers to eliminate congestion by running at 1 Gbps. All calibration experiments were run with drop-tail queues having 2,400 queue elements (the reasons for this choice are discussed in Section 4). We ran one instance of the browser program on each of the browser machines and one instance of the server program on all the server machines. Each browser was configured to emulate the same number of active users and the total active users varied from 7,000 to 35,000 over several experiments. Figure 2 shows the aggregate traffic on one direction of the 1 Gbps link as a function of the number of emulated users. The load in the opposite direction was measured to be essentially the same and is not plotted in this figure. The offered load expressed as link throughput is a linear function of the number of emulated users indicating there are no fundamental resource limitations in the system and generated loads can easily exceed the capacity of a 100 Mbps link. With these data we can determine the number of emulated users that would generate a specific offered load if there were no bottleneck link present. This capability is used in subsequent experiments to control the offered loads on the network. For example, if we want to generate an offered load equal to the capacity of a 100 Mbps link, we use Figure 2 to determine that we need to emulate approximately 19,040 users (9,520 on each side of the link). Note that for offered loads approaching saturation of the 100 Mbps link, the actual link utilization will, in general, be less than the intended offered load. This is because as response times become longer, users have to wait longer before they can generate new requests and hence generate fewer requests per unit time.

A motivation for using Web-like traffic in our experiments was the assumption that properly generated traffic would exhibit demands on the laboratory network consistent with those found in empirical studies of real networks, specifically, a long-range dependent (LRD) packet arrival process. The empirical data used to generate our web traffic showed heavy-tailed distributions for both user "think" times and response sizes [13]. (Figures 3-4 compare the cumulative distribution function (CDF), $F(x) = \Pr[X \le x]$, and complementary cumulative distribution function (CCDF), $1 - F(x)$, of the generated responses in the calibration experiments with the empirical distribution from [13]. Note from Figure 3 that while the median response size in our simulations will be approximately 1,000 bytes, responses as large as 10^9 bytes will also be generated.)

That our web traffic showed heavy-tailed distributions for both think times (OFF times) and response size (ON times), implies that the aggregate traffic generated by our large collection of sources should be LRD [14]. To verify that such LRD behavior is indeed realized with our experimental setup, we recorded *tcpdumps* of all TCP/IP headers during the calibration experiments and derived a time series of the number of packets and bytes arriving on the 1 Gbps link between the routers in 1 millisecond time intervals. We used this time series with a number of analysis methods (aggregated variance, Whittle, Wavelets) to estimate the Hurst parameter. In all cases the 95% confidence intervals for the estimates fell in the range 0.8 to 0.9 which indicates a significant LRD component in the time series.

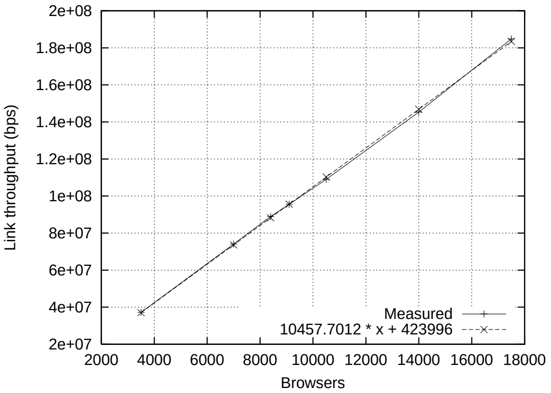

Figure 2: Link throughput *v*. number of emulated browsing users compared to a straight line.

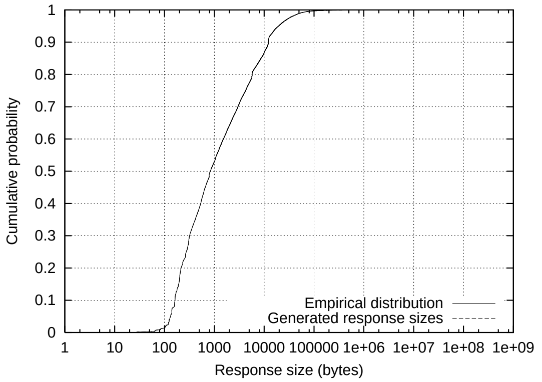

Figure 3: CDF of empirical *v*. generated response sizes.

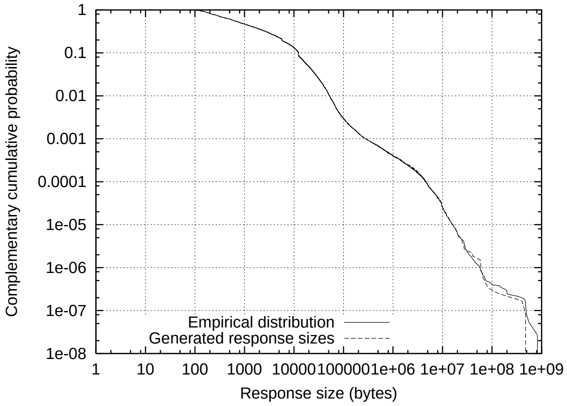

Figure 4: CCDF of empirical *v*. generated response sizes.

3.3 Experimental Procedures

Each experiment was run using the following automated procedures. After initializing and configuring all router and end-system parameters, the server programs were started followed by the browser programs. Each browser program emulated an equal number of users chosen, as described above, to place a nominal offered load on an unconstrained network. The offered loads used in the experiments were chosen to represent user populations that could consume 80%, 90%, 98%, or 105% of the capacity of the 100 Mbps link connecting the two router machines (*i.e.*, consume 80, 90, 98 or 105 Mbps, respectively). It is important to emphasize again that terms like "105% load" are used as a shorthand notation for "a population of web users that would generate a long-term average load of 105 Mbps on a 1 Gbps link." Each experiment was run for 120 minutes to ensure very large samples (over 10,000,000 request/response exchanges in each experiment) but data were collected only during a 90-minute interval to eliminate startup effects at the beginning and termination synchronization anomalies at the end. Each experiment for a given AQM schemes was repeated three times with a different set of random number seeds for each repetition. To facilitate comparisons among different AQM schemes, experiments for different schemes were run with the same sets of initial seeds for each random number generator (both those in the traffic generators for sampling various random variables and in *dummynet* for sampling minimum per-flow delays).

The key indicator of performance we use in reporting our results are the end-to-end response times for each request/response pair. We report these as plots of the cumulative distributions of response times up to 2 seconds.[2] In these plots we show only the results from one of the three repetitions for each experiment (usually there were not noticeable differences between repetitions; where there were, we always selected the one experiment most favorable to the AQM scheme under consideration for these plots). We also report 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. These results are reported in Table 2 where the values shown are means over the three repetitions of an experiment.

4 AQM EXPERIMENTS WITH PACKET DROPS

For both PI and REM we chose two target queue lengths to evaluate: 24 and 240 packets. These were chosen to provide two operating points: one that potentially yields minimum latency (24) and one that potentially provides high link utilization (240). The values used for the coefficients in the control equations above are those recommended in [1, 8] and confirmed by the algorithm designers. For ARED we chose the same two target queue lengths to evaluate. The calculations for all the ARED parameter settings follow the guidelines given in [6] for achieving the desired target delay (queue size). In all three cases we set the maximum queue size to a number of packets sufficient to ensure tail drops do not occur.

To establish a baseline for evaluating the effects of using various AQM designs, we use the results from a conventional drop-tail FIFO queue. In addition to baseline results for drop-tail at the queue sizes 24 and 240 chosen for AQM, we also attempted to find a queue size for drop-tail that would represent a "best practice" choice. Guidelines (or "rules of thumb") for determining the "best" allocations of queue size have been widely debated in various venues including the IRTF *end2end-interest* mailing list. One guideline that appears to have attracted a rough consensus is to provide buffering approximately equal to 2-4 times the bandwidth-delay product of the link. Bandwidth in this expression is that of the link and the delay is the mean round-trip time for all connections sharing the link — a value that is, in general, difficult to determine. Other mailing list contributors have recently tended to favor buffering equivalent to 100 milliseconds at the link's transmission speed. FreeBSD queues are allocated in terms of a number of buffer elements (*mbufs*) each with capacity to hold an IP datagram of Ethernet MTU size. For our experimental environment where the link bandwidth is 100 Mbps and the mean frame size is a little over 500 bytes, this implies that a FIFO queue should have available about 2,400 mbufs for 100 milliseconds of buffering.

Figures 5-7 give the response-time performance of a drop-tail queue with 24, 240 and 2,400 queue elements for offered loads of 80%, 90%, and 98% compared to the performance on the un-congested 1 Gbps link. Loss rates and link utilizations are given in Table 2. At 80% load (80 Mbps on a 100 Mbps link) the results for any queue size are indistinguishable from the results on the un-congested link. At 90% load we see some significant degradation in response times for all queue sizes but note that, as expected, a queue size of 24 or 240 elements is superior for responses that are small enough to complete in under 500 milliseconds. For the longer queue of 2,400 elements performance is somewhat better for the longer responses. At a load of 98% there is a severe performance penalty to response times but, clearly, a shorter queue of 240 elements is more desirable than one of 2,400 elements. In Figure 7 we also see a feature that is found in all our results at high loads where there are significant numbers of dropped packets (see Table 2). The flat area in the curves for 24 and 240 queue sizes shows the impact of RTO granularity in TCP — most responses with a timeout take at least 1 second. In this study we made no attempt to empirically determine the "optimal" queue size for the drop-tail queue in our routers (which is likely to be somewhere between 300 and 2,000 elements). Finding such a drop-tail queue size involves a tradeoff between improving response times for the very large number of small objects versus the small number of very large objects that consume most of the network's capacity. Instead, we use a drop-tail queue of 240 elements as a baseline for comparing with AQM mechanisms because it corresponds to one of the targets selected for AQM and provides reasonable performance for drop-tail.

4.1 Results for PI with Packet Drops

Figure 8 gives the results for PI at target queue lengths of 24 and 240, and offered loads of 80%, 90%, and 98%. At 80% load, there is essentially no difference in response times between the two target queue lengths, and their performance is very close to that obtained on the un-congested network. At 90% load, the two target queue sizes provide nearly identical results except for the 10% of responses requiring more than 500 milliseconds to complete. For these latter requests, the longer target size of 240 is somewhat better. At 98% load, the shorter queue target of 24 is a better choice as it improves response times for shorter responses but does not degrade response times for the longer responses.

4.2 Results for REM with Packet Drops

Figure 9 gives the results for REM at target queue lengths of 24 and 240, and offered loads of 80%, 90%, and 98%. At 80% load, there is essentially no difference in response times between the two target queue lengths, and their performance is very close to that obtained on the un-congested network. At 90% load, a queue reference of 24 performs much better than a target queue of 240. At 98% load, a queue reference of 24 continues to perform slightly better than 240. Overall, REM performs best when used with a target queue reference of 24.

[2] Because of space restrictions, only plots of summary results are shown for105% load.

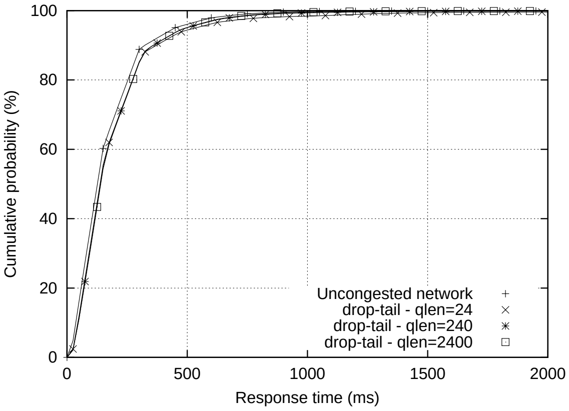

Figure 5: Drop-tail performance, 80% offered load.

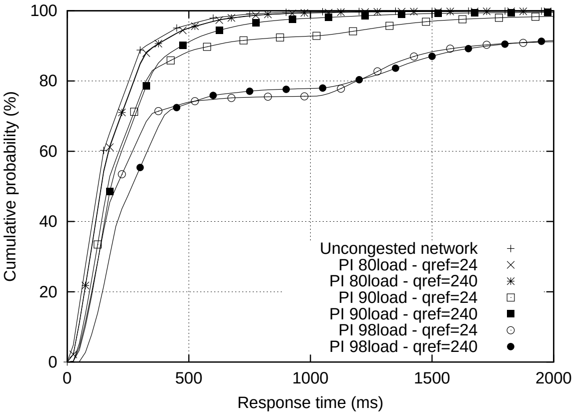

Figure 8: PI performance with packet drops.

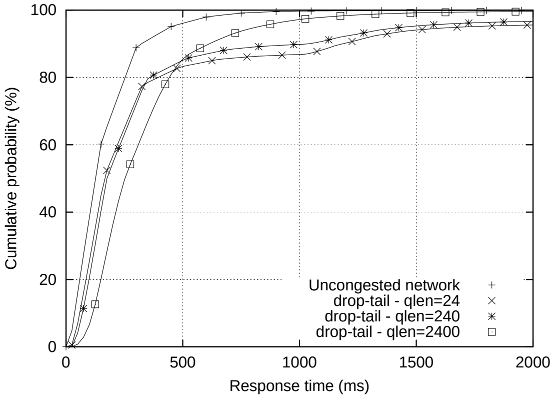

Figure 6: Drop-tail performance, 90% offered load.

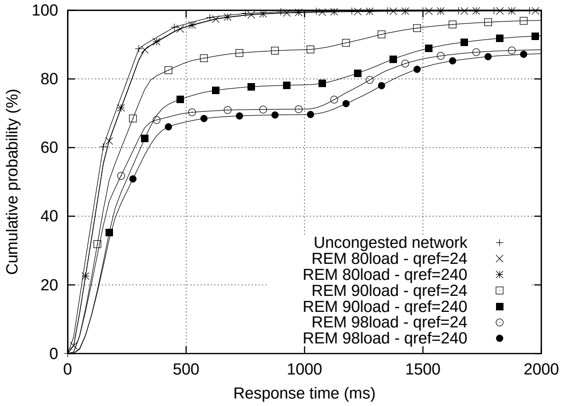

Figure 9: REM performance with packet drops.

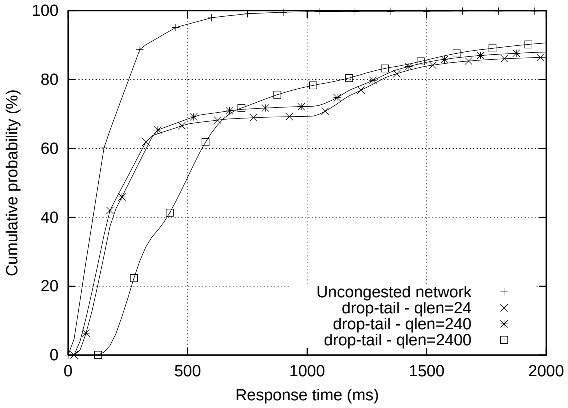

Figure 7: Drop-tail performance, 98% offered load.

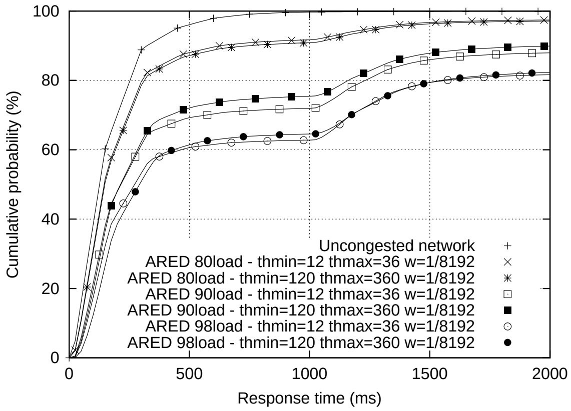

Figure 10: ARED performance with packet drops.

4.3 Results for ARED with Packet Drops

Figure 10 gives the results for ARED at the two target queue lengths, and offered loads of 80%, 90%, and 98%. These results are all quite surprising. In contrast to drop-tail, PI, and REM, the results at 80% load show some degradation relative to the results on the un-congested link. At this load there is essentially no difference in response times between the two target queue lengths. At 90% load, there is still little difference in performance among the two target queues for ARED but the overall degradation from an un-congested link is more substantial. At 98% load, the two settings for ARED are again almost indistinguishable from each other and response times, overall, are very poor.

Because of the consistently poor performance of ARED, we tried several different sets of parameters. We tried the recommended settings for our network capacity of min_{th} at 60, max_{th} at 180 and w_q at 1/16384. We also tried varying the parameter w_q from 1/1024 up to 1/16384, but none of the settings yielded results that were better than the ones presented here. We cannot recommend a setting for ARED based on our experiments since the performance for all of them are very close to each other, and yet, unsatisfactory.

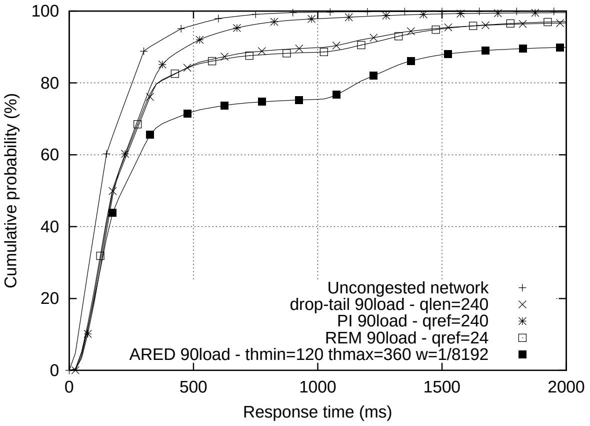

Figure 11: Comparison of all schemes at 90% load.

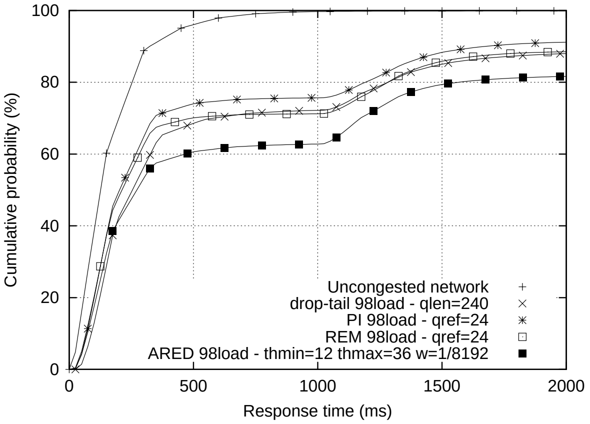

Figure 12: Comparison of all schemes at 98% load.

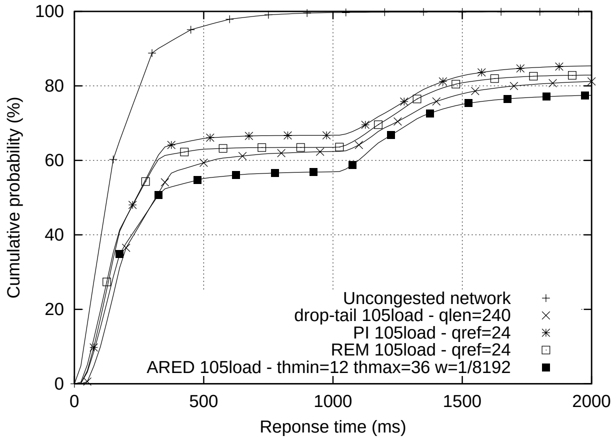

Figure 13: Comparison of all schemes at 105% load.

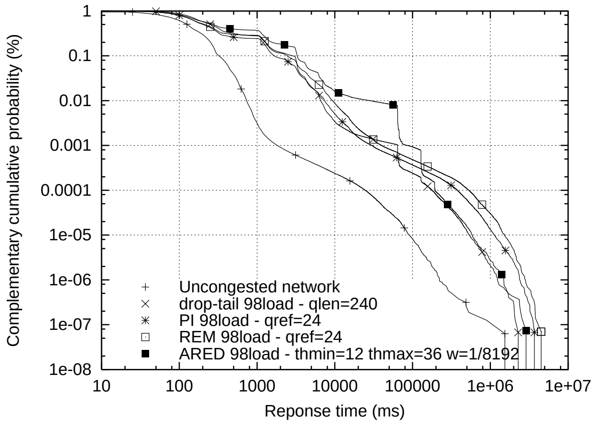

Figure 14: CCDF of all schemes without ECN, 98% load.

4.4 Comparing all Schemes with Packet Drops

At 80% load, all schemes but ARED perform comparably to an uncongested network, and are barely distinguishable from each other. Figures 11-14 compare the best settings, based on the overall distribution of response times, for each AQM scheme for offered loads of 90%, 98%, and 105%. In comparing results for two AQM schemes, we claim that the response time performance is better for one of them if its CDF is clearly above the other's in some substantial range of response times and comparable in the remaining range. At 90% load, PI, REM, and drop-tail all provide reasonable performance for the 80% of responses that can be completed in 400 milliseconds or less. For the remaining 20% of responses, PI with a target queue length of 240 is better than the other schemes. Overall, PI with a target queue of 240 provides very good performance at this load. At 98% load, PI is again somewhat superior to the other schemes but note that the best performance is obtained with a target queue length of 24 and that overall, no AQM scheme can offset the performance degradation at this extreme load. At 105% load performance for all schemes degrades uniformly from the 98% case. Table 2 also presents the link utilization, loss ratios, and the number of completed requests for each experiment for each AQM scheme. At 90% and 98% offered loads, drop-tail with a queue of 240 gives slightly better link utilization than any of the AQM schemes. It also completes slightly more request-response exchanges than the other schemes at the same load. Drop-tail does, however, have higher loss ratios than the other schemes. PI has better loss ratios than REM, completes more requests, and has better link utilization at all loads.

Figures 11-12 show that at least 90% of all responses complete in under 2 seconds for the best AQM schemes. Figure 14 shows the remainder of the distribution at 98% load. The conclusions drawn from Figures 11-13 also hold for responses that experience response times up to approximately 50 seconds (~99.95% of all responses). The remaining responses perform best under drop-tail. Eventually ARED performance approaches that of drop-tail and is superior to PI and REM but only for a handful of responses.

5 AQM EXPERIMENTS WITH ECN

AQM schemes drop packets as an indirect means of signaling congestion to end-systems. The explicit congestion notification (ECN) packet marking scheme was developed as a means of explicitly signaling congestion to end-systems [12]. To signal congestion a router can "mark" a packet by setting a specified bit in the TCP/IP header of the packet. This marking is not modified by subsequent routers. Upon receipt of a marked data segment, a TCP receiver will mark the TCP header of its next outbound segment (typically an ACK) destined for the sender of the original marked segment. Upon receipt of this marked segment, the original sender will react as if a single segment had been lost within a send window. In addition, the sender will mark its next outbound segment (with a different marking) to confirm that it has reacted to the congestion.

Table 2: Loss, completed requests, and link utilizations.

	Offered Load	Loss ratio (%)		Completed requests (millions)		Link utilization/ throughput (Mbps)	
		No ECN	ECN	No ECN	ECN	No ECN	ECN
Uncongested 1 Gbps network (drop-tail)	80%	0		13.2		80.6	
	90%	0		15.0		91.3	
	98%	0		16.2		98.2	
	105%	0		17.3		105.9	
drop-tail queue size = 24	80%	0.2		13.2		80.3	
	90%	2.7		14.4		88.4	
	98%	6.5		14.9		91.1	
	105%	9.1		15.0		91.8	
drop-tail queue size = 240	80%	0.04		13.2		80.6	
	90%	1.8		14.6		89.9	
	98%	6.0		15.1		92.0	
	105%	8.8		15.0		92.4	
drop-tail queue size = 2,400	80%	0		13.1		80.4	
	90%	0.1		14.7		88.6	
	98%	3.6		15.1		91.3	
	105%	7.9		15.0		91.1	
PI $q_{ref} = 24$	80%	0	0	13.3	13.2	80.2	79.3
	90%	1.3	0.3	14.4	14.6	87.9	88.6
	98%	3.9	1.8	15.1	14.9	89.3	89.4
	105%	6.5	2.5	15.1	15.0	89.9	89.5
PI $q_{ref} = 240$	80%	0	0	13.1	13.1	80.1	80.1
	90%	0.1	0.1	14.7	14.7	87.2	88.2
	98%	3.7	1.7	14.9	15.1	90.0	89.6
	105%	6.9	2.3	15.0	15.2	90.5	90.8
REM $q_{ref} = 24$	80%	0.01	0	13.2	13.1	79.8	80.1
	90%	1.8	0.1	14.4	14.6	86.4	88.2
	98%	5.0	1.7	14.5	14.9	87.6	89.6
	105%	7.7	2.4	14.6	14.9	87.5	89.3
REM $q_{ref} = 240$	80%	0	0	13.2	13.2	79.3	80.3
	90%	3.3	0.2	14.0	14.7	83.3	88.6
	98%	5.4	1.6	14.4	15.1	86.2	90.4
	105%	7.3	2.3	14.6	15.1	87.7	90.4
ARED $th_{min} = 12$ $th_{max} = 36$ $w_q = 1/8192$	80%	0.02	0.03	13.0	12.9	79.4	78.8
	90%	1.5	1.3	13.8	13.8	85.5	85.5
	98%	4.1	4.1	14.0	13.9	87.4	88.0
	105%	5.1	5.1	14.1	14.1	87.3	87.7
ARED $th_{min} = 120$ $th_{max} = 360$ $w_q = 1/8192$	80%	0.02	0.02	13.0	13.1	80.2	80.5
	90%	1.4	1.2	14.0	14.1	85.5	86.2
	98%	4.8	4.7	14.2	14.1	87.9	88.2
	105%	6.8	6.3	13.9	13.9	85.2	85.8

We repeated each of the above experiments with PI, REM, and ARED using packet marking and ECN instead of packet drops. Up to 80% offered load, ECN has no effect on response times of any of the AQM schemes. Figures 15-20 show the results for loads of 90% and 98%. At 90% load, with target queue length of 24, PI performs better with ECN, however, with a target queue length of 240, there is little change in performance. At 98% load, ECN significantly improves performance for PI at both target queue lengths. REM shows the most significant improvement in performance with ECN. Although PI performed better than REM when used without ECN at almost all loads, at 90% and 98% loads PI and REM with ECN give very similar performance, and ECN has a significant effect on PI and REM performance in almost all cases. Overall, and contrary to the PI and REM results, ECN has very little effect on the performance of ARED at all tested target queue lengths at all loads.

Table 2 again presents the link utilization, loss ratios, and the number of completed requests for each ECN experiment. PI with ECN clearly seems to have better loss ratios, although there is little difference in link utilization and number of requests completed. REM's improvement when ECN is used derives from lowered loss ratios, increases in link utilization, and increases in number of completed requests. With ARED, there is very little improvement in link utilization or number of completed requests. Its loss ratios are only marginally better with ECN.

5.1 Comparisons of PI, REM, & ARED with ECN

Recall that at 80% load, no AQM scheme provides better response time performance than a simple drop-tail queue. This result is not changed by the addition of ECN. Here we compare the best settings for PI, REM, and ARED when combined with ECN for loads of 90%, 98%, and 105%. The results for drop-tail (queue length 240) are also included as a baseline for comparison. Figures 21-23 show these results. At 90% load, both PI and REM provide response time performance that is both surprisingly close to that on an un-congested link and is better than drop-tail. At 98% load there is noticeable response time degradation with either PI or REM, however, the results are far superior to those obtained with drop-tail. Further, both PI and REM combined with ECN have substantially lower packet loss rates than drop-tail and link utilizations that are only modestly lower. At 105% load the performance of PI and REM is virtually identical and only marginally worse than was observed at 98% load. This is an artifact of our traffic generation model wherein browsers generate requests less frequently as response times increase. Table 2 shows that few additional request-response exchanges are completed at 105% load than at 98% load. For ARED, even when used with ECN, response time performance at all load levels is significantly worse than PI and REM except for the shortest 40% of responses where performance is comparable.

Figure 24 shows the tails of the response time distribution at 98% load. For AQM with ECN, drop-tail again eventually provides better response time performance, however, the crossover point occurs earlier, at approximately 5 seconds. The 1% of responses experiencing response times longer than 5 seconds receive better performance under drop-tail. ARED performance again eventually approaches that of drop-tail for a handful of responses.

6 DISCUSSION

Our experiments have demonstrated several interesting differences in the performance of Web browsing traffic under control theoretic and pure random-dropping AQM. Most striking is the response time performance achieved under PI and REM with ECN at loads of 90% and 98%. In particular, at 90% load response time performance surprisingly approximates that achieved on an un-congested network. Approximately 90% of all responses complete in 500 milliseconds or less whereas only approximately 95% of responses complete within the same threshold on the un-congested network.

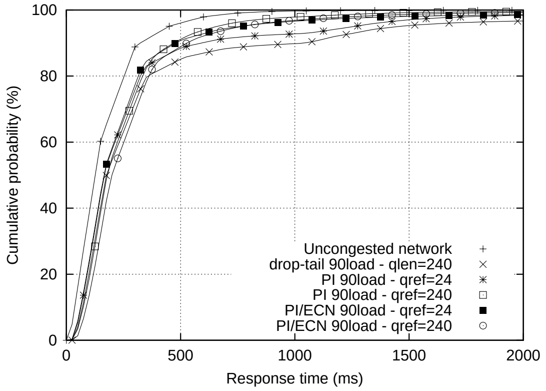

Figure 15: PI performance with/without ECN, 90% load.

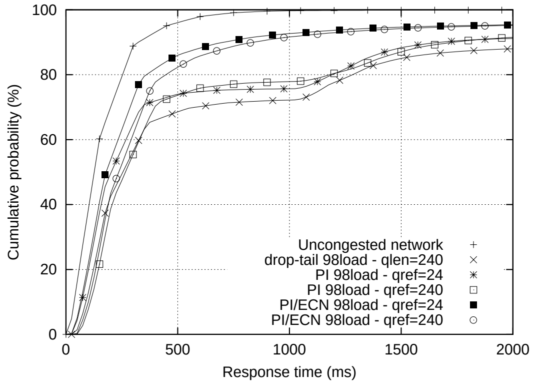

Figure 16: PI performance with/without ECN, 98% load.

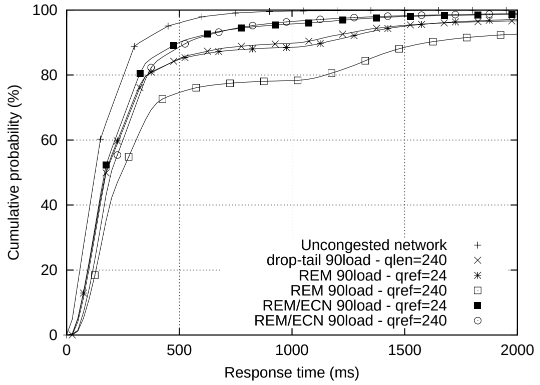

Figure 17: REM performance with/without ECN, 90% load.

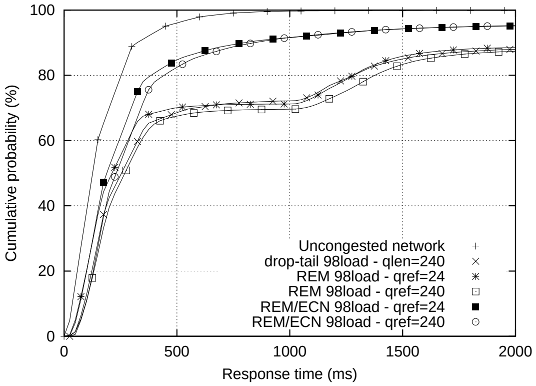

Figure 18: REM performance with/without ECN, 98% load.

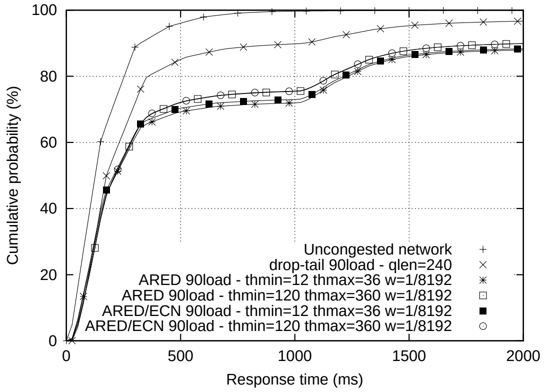

Figure 19: ARED performance with/without ECN, 90% load.

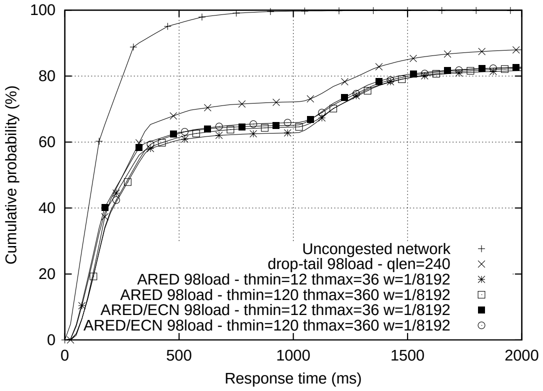

Figure 20: ARED performance with/without ECN, 98% load.

To better understand PI's distributions of response times and the positive impact of ECN, Figures 25-26 show scatter plots of response size versus response time for PI at 98% load. (In interpreting these plots it is important to remember that the median response size is under 1,000 bytes and the 90th percentile response is slightly over 10,000 bytes (see Figure 3).) For small responses, strong banding effects are seen at multiples of 1 second representing the effects of timeouts. Of special interest is the density of the band at 6 seconds representing the effects of a dropped SYN segment. While it appears that PI forces a large number of small responses to experience multi-second response times, PI in fact does better in this regard than all the other AQM schemes. With the addition of ECN, the number of timeouts is greatly reduced and PI enables the vast majority of all responses to experience response times proportional to their RTT divided by their window size. This is seen in Figure 26 by observing the dense triangle-shaped mass of points starting at the origin and extending outward to the points (100,000, 6,000) and (100,000, 500). (Note as well the existence of similar triangles offset vertically by multiples of 1 second — the canonical packet loss timeout.)

The second most striking result is that performance varied substantially between PI and REM with packet dropping and this performance gap was closed through the addition of ECN. A preliminary analysis of REM's behavior suggests that ECN is not so much improving REM's behavior as it is ameliorating a fundamental design problem. Without ECN, REM consistently causes flows to

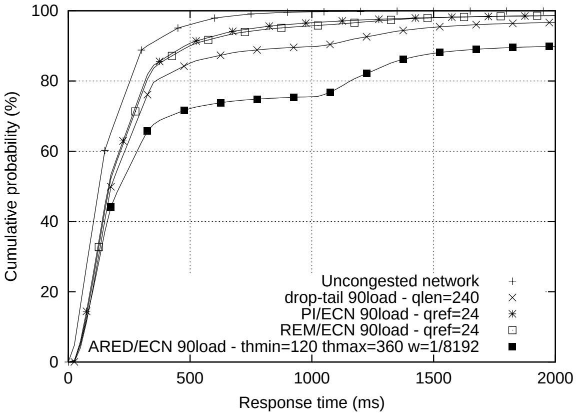

Figure 21: Comparison of all schemes with ECN, 90% load.

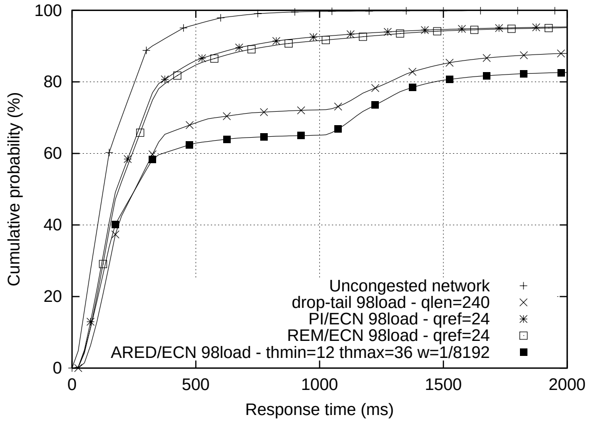

Figure 22: Comparison of all schemes with ECN, 98% load.

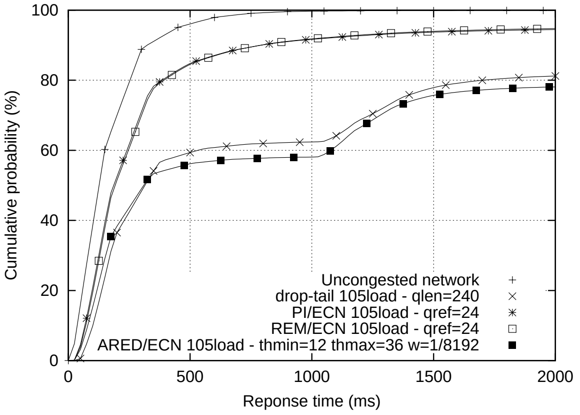

Figure 23: Comparison of all schemes with ECN, 105% load.

experience multiple drops within a source's congestion window, forcing flows more frequently to recover the loss through TCP's timeout mechanism rather than its fast recovery mechanism. When ECN is used, REM simply marks packets and hence even if multiple packets from a flow are marked within a window the timeout will be avoided. Thus ECN appears to improve REM's performance by mitigating the effects of its otherwise poor (compared to PI) marking/dropping decisions.

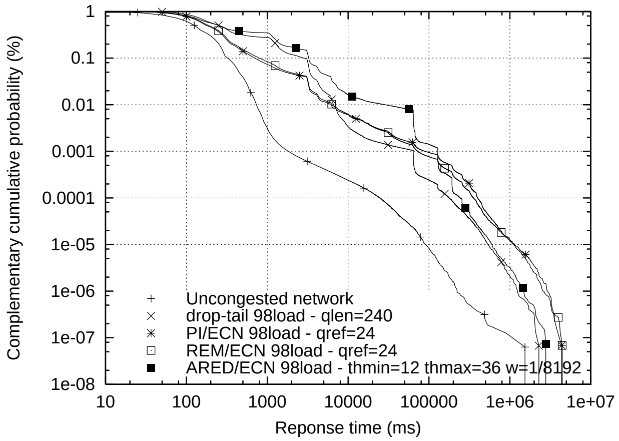

Figure 24: CCDF of all schemes with ECN, 98% load.

The final point of note is the difference in performance between ARED and the other AQM schemes, in particular, the fact that response time performance is consistently worse with ARED than with drop-tail. The exact reasons for the observed differences remains the subject of continued study, however, the experiments reported here and others lead us to speculate that there are three primary factors influencing these results. First, PI and REM operate in "byte mode" by default — they monitor the queue length in bytes rather than packets. While ARED also has a byte mode, "packet mode" is the recommended setting. Byte mode allows for finer grain queue measurements but more importantly, in PI and REM the marking/dropping probability for an individual packet is biased by a factor equal to the ratio of the current packet size to the average (or maximum) packet size. This means that in PI and REM, SYNs and pure ACKs experience a lower drop probability than data segments arriving at the router at the same time and hence fewer SYNs and ACKs are dropped than under ARED.

Second, in ARED's "gentle mode," when the average queue size is between max_{th} and 2xmax_{th}, ARED drops ECN-marked packets, following the ECN guidelines that state packets should be dropped when the AQM scheme's queue length threshold is exceeded in this manner. The motivation for this rule is to more effectively deal with potential non-responsive flows that are ignoring congestion indications [12]. Our analysis indicates that this rule is in fact counter-productive and explains much of ARED's inability to benefit from ECN.

Finally, PI and REM periodically sample the (instantaneous) queue length when deciding to mark packets. ARED uses a weighted average. We believe that the reliance on the average queue length significantly limits ARED's ability to react effectively in the face of highly bursty traffic such as the Web traffic generated herein. Of note was the fact that changing ARED's weighting factor for computing average queue length by an order of magnitude had no effect on performance.

7 CONCLUSIONS

From the results reported above we draw the following conclusions. These conclusions are based on a premise that user-perceived response times are the primary yardstick of performance and that link utilization and packet loss rates are important but secondary measures.

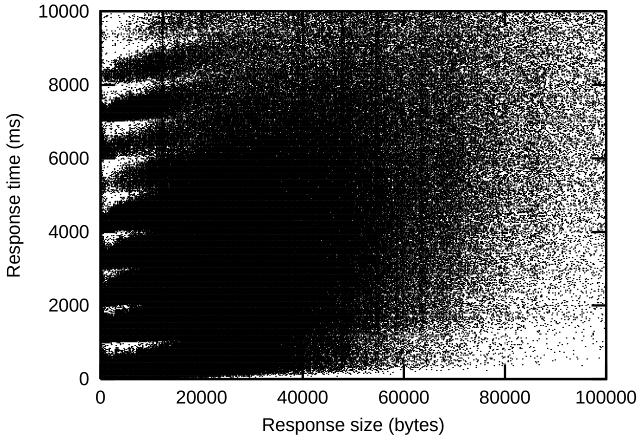

Figure 25: Scatter plot of PI performance without ECN, 98% load.

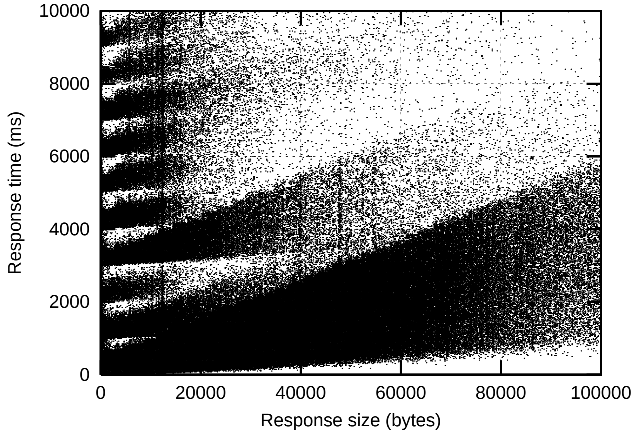

Figure 26: Scatter plot of PI performance with ECN, 98% load.

- For offered loads up to 80% of bottleneck link capacity, no AQM scheme provides better response time performance than simple drop-tail FIFO queue management. Further, the response times achieved on a 100Mbps link are not substantially different from the response times on a 1 Gbps link with the same number of active users that generate this load. This result is not changed by combining any of the AQM schemes with ECN.
- For loads of 90% of link capacity or greater, PI results in a modest improvement over drop-tail and the other AQM schemes when ECN is not used.
- With ECN, both PI and REM provide significant response time improvement at offered loads at or above 90% of link capacity. Moreover, at a load of 90%, PI and REM with ECN provide performance on a 100 Mbps link competitive with that achieved with a 1 Gbps link with the same number of active users.
- ARED with recommended parameter settings consistently resulted in the poorest response time performance. This result was not changed with the addition of ECN.

We conclude that without ECN there is little end-user performance gain to be realized by employing any of the AQM schemes studied here. However, with ECN performance can be significantly improved at near-saturation loads with either PI or REM. Thus, it appears likely that provider links may be operated at 80% of capacity even when not deploying any AQM (with or without ECN). Further, providers may be able to operate their links at even higher load levels without significant degradation in user-perceived performance provided PI or REM combined with ECN is deployed in their routers and ECN is implemented in TCP/IP stacks on the end-systems.

8 ACKNOWLEDGEMENTS

We are indebted to Sanjeewa Athuraliya, Sally Floyd, Steven Low, Vishal Misra, and Don Towsley, for their assistance in performing the experiments described herein. In addition, we are grateful for the constructive comments of the anonymous referees and for the help of our shepherd, Dina Katabi.

This work was supported in parts by the National Science Foundation (grants ITR-0082870, CCR-0208924, and EIA-0303590), Cisco Systems Inc., and the IBM Corporation.

9 REFERENCES

[1] S. Athuraliya, A Note on Parameter Values of REM with Reno-like Algorithms, *http://netlab.caltech.edu*, March 2002.

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

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

[4] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith, *Tuning RED for Web Traffic*, Proc., ACM SIGCOMM 2000, Sept. 2000, pp. 139-150.

[5] W. Feng, D. Kandlur, D. Saha, K. Shin, *A Self-Configuring RED Gateway*, Proc., INFOCOM '99, March 1999, pp. 1320-1328.

[6] 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.

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

[8] C.V. Hollot, V. Misra, W.-B. Gong, D. Towsley, *On Designing Improved Controllers for AQM Routers Supporting TCP Flows*, Proc., IEEE INFOCOM 2001, April 2001, pp. 1726-1734

[9] L. Rizzo, *Dummynet: A simple approach to the evaluation of network protocols*, ACM CCR, Vol. 27, No. 1, January 1997, pp. 31-41.

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

[11] V. Misra, W.-B. Gong,, D. Towsley, *Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED*, Proc., ACM SIGCOMM 2000, pp. 151-160.

[12] K. Ramakrishnan, S. Floyd, D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168, September 2001.

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

[14] W. Willinger, M.S. Taqqu, R. Sherman, D. Wilson, *Self-similarity through high variability: statistical analysis of ethernet LAN traffic at the source level*, IEEE/ACM Transactions on Networking, Vol. 5, No. 1, February 1997, pp. 71-86.