Tuning RED for Web Traffic*

*Mikkel Christiansen,*** *Kevin Jeffay,* *David Ott,* *F. Donelson Smith*
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175 USA
http://www.cs.unc.edu/Research/dirt

Abstract

We study the effects of RED on the performance of Web browsing with a novel aspect of our work being the use of a user-centric measure of performance — response time for HTTP request-response pairs. We empirically evaluate RED across a range of parameter settings and offered loads. Our results show that: (1) contrary to expectations, compared to a FIFO queue, RED has a minimal effect on HTTP response times for offered loads up to 90% of link capacity, (2) response times at loads in this range are not substantially effected by RED parameters, (3) between 90% and 100% load, RED can be carefully tuned to yield performance somewhat superior to FIFO, however, response times are quite sensitive to the actual RED parameter values selected, and (4) in such heavily congested networks, RED parameters that provide the best link utilization produce poorer response times. We conclude that for links carrying only web traffic, RED queue management appears to provide no clear advantage over tail-drop FIFO for end-user response times.

1. Introduction

A recent IETF publication strongly recommended the widespread deployment of active queue management technology in routers to improve the performance of today's Internet [4]. Active queue management refers to the practice of manipulating the queue at an outbound interface in a router to bias the performance of flows that transit the router. The goals of active queue management are to (1) reduce the average length of queues in routers and thereby decrease the end-to-end delay experienced by packets, and (2) ensure that network resources are used more efficiently by reducing the packet loss that occurs when queues overflow.

The recommended active queue management to be deployed is *random early detection*, better known as RED [13]. Under RED, a router will probabilistically drop an arriving packet even though the queue for the appropriate outbound interface is not full. The motivation for this "early" drop comes from the fact that packet loss is the primary indicator of congestion for a TCP connection. By dropping packets before a router's queue fills, the TCP connections sharing the queue will reduce their transmission rates and (ideally) ensure the queue does not overflow. The claim (borne out by significant empirical data) is that dropping packets prior to the overflow of the queue will reduce the overall rate of packet loss. Given that TCP traffic dominates on Internet backbones [25], RED, and other forms of early congestion notification, have the potential to improve overall network performance as well as that seen by individual TCP connections. In this work we test this claim and explore the impact of RED on the performance of the most dominant subset of TCP connections on the Internet today: Web traffic. In particular, we are interested in measuring the effect of RED on a user-centric measure of performance — the response time for an HTTP 1.0 request. Although the performance of RED and other early congestion notification mechanisms continue to be the subject of much study, the evaluation metrics have largely been network-centric measures such as network link utilization or aggregate TCP throughput. Moreover, as argued in Section 2 below, most of these evaluation studies focused on simulations of long-lived TCP connections such as (huge) file transfers. In contrast, measurement studies have shown that the majority of TCP connections are HTTP connections[1] and that many of these connections are quite short-lived, often on the order of a few TCP segments. More importantly, given that the performance of the Internet is becoming synonymous with the performance of the Web, understanding the impact of router forwarding behaviors on user-visible performance measures is an important (and largely ignored) aspect of the evaluation of any congestion control proposal.

At a high-level, we seek to compare the performance of HTTP request-response pairs under RED and more traditional tail-drop[2] FIFO queuing. Unfortunately, measuring the performance of HTTP under RED is a complex problem. First, as described in more detail in Section 3, RED is a general mechanism that is controlled by (at least) 5 separate control parameters. There exist rules-of-thumb for assigning values to most parameters [15], but little is known about how (or if) one can optimize RED performance for a given traffic class. Second, even if optimal RED parameter settings were known, generating or simulating HTTP behaviors in a meaningful way is problematic. There are few models of HTTP traffic and it is likely the case that Web traffic dynamics (*e.g.*, the mix between HTTP 1.0 and 1.1 protocols) are evolving faster than our current ability to measure and model the traffic.

Our general approach is to conduct a "live simulation" of Web browsing in a laboratory environment. By live simulation, we mean that we simulate a large collection of users browsing the Web at a set of sites distributed throughout the continental United States. The HTTP traffic generated by the simulated users will traverse a laboratory network with routers that support both RED and traditional tail-drop FIFO queuing. A number of instances of the user-browsing simulation program are

* Work supported by grants from the National Science Foundation (grants CDA-9624662 and ITR 00-82870), Cisco Systems, the Intel Corp., and the North Carolina Networking Initiative.

** Current address: Aalborg University, Department of Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark. This work was performed while the first author was a visiting student at the University of North Carolina.

[1] For example, recent measurements on the MCI backbone show that about 95% of the bytes transmitted across the network are carried by TCP and of these, 50-70% are HTTP messages [25].

[2] Tail-drop refers to the action of dropping a packet that was intended to be added at the tail of a full FIFO queue.

Published in: *IEEE/ACM Transactions on Networking*, Volume 9, Number 3, (June 2001), pages 249-264.

run to generate a configurable offered load on a bottleneck network link. The user HTTP requests will be delivered to a set of servers that will respond with responses of the appropriate sizes. Both request and response packets are artificially delayed to simulate the round trip times (RTT) experienced when communicating with machines distributed across the US. This is done to ensure our end-to-end response-time measurements reflect the full range of effects of TCP congestion control and retransmissions experienced by real users. When the responses are delivered back to the users, we record the elapsed time for each simulated HTTP request/response pair.

This experimental setup provides a basis for comparing the effect of RED *v.* FIFO queuing on the response time for HTTP requests. We performed a series of experiments to empirically determine the FIFO queue length and combination of RED parameter settings that result in the best performance for our network and our simulation of Web traffic. From our experiments we observe the following:

- Contrary to expectations, when compared to a (properly configured) tail-drop FIFO queue, RED has a minimal effect on HTTP response times for offered loads up to 90% of link capacity.
- Response times for loads in this range are not substantially effected by values of RED parameters.
- Between loads of 90% to 100% of link capacity, RED can be carefully tuned to yield performance somewhat superior to FIFO. However, response times are quite sensitive to the actual RED parameter values selected. In our experiments recommended parameter settings resulted in poorer performance than FIFO. Worse, the "optimal" settings that resulted in the best RED performance were non-obvious and arrived at only through exhaustive trial-and-error experimentation.
- For loads of 90% to 100% of link capacity where RED has the potential to provide better performance, performance becomes a subjective measure. For loads in this range there exists a trade-off between improving response times of short-lived connections and improving response times of long-lived connections. Both cannot be optimized simultaneously.
- In such heavily congested networks, there exists a trade-off between network utilization and HTTP transaction response times. RED parameters values that provide the best link utilization produce poorer response times.

We have considered only HTTP traffic in our experiments and hence our results are best interpreted as representing a worst-case scenario for RED performance on real Internet links that carry a mix of HTTP and other traffic classes. Nonetheless, we conclude that for links carrying only web traffic, RED appears to provide no clear advantage over tail-drop FIFO for end-users whose primary metric of satisfaction is response time. Moreover, given the lack of engineering practice to guide the setting of RED parameter values, and our demonstration that "reasonable," but nonetheless sub-optimal RED parameters values can result in poorer performance than FIFO queuing, without further analysis it is possible that widespread RED deployment may not provide the expected benefits.

The remainder of this paper is organized as follows. Section 2 provides a more in-depth introduction to RED and reviews the literature in the performance evaluation of RED and related active queue management schemes. Section 3 describes our experimental methods and the design and calibration of our experiments. Section 4 presents the performance of our simulated Web browsing sessions under FIFO queuing; Section 5 presents results for RED queuing. Section 6 provides a preliminary analysis of the observed response time distributions under RED. Section 7 compares the RED and FIFO results. We conclude in Section 8 with a discussion of the results, the limitations of our experiments and results, and some comments on future work.

For completeness, we include two appendices. Appendix A provides additional details on our experimental setup and methods. Appendix B provides additional result data on our RED and FIFO experiments.

2. Background and Related Work

RED is an active queue management mechanism that is intended to address some of the shortcomings of standard tail-drop FIFO queue management [4]. In a FIFO queue it is possible for "lock-out" to occur, a condition in which a small subset of the flows sharing the link can monopolize the queue during periods of congestion. Flows generating packets at a high rate can fill up the queue such that packets from flows generating packets at substantially lower rates have a higher probability of arriving at the queue when it is full and being discarded. A second problem with a FIFO queue is that latency is increased for all flows when the queue is constantly full. Simply making the queue shorter will decrease the latency but negates the possibility of accommodating brief bursts of traffic without dropping packets unnecessarily. RED addresses both the "lock-out" problem by using a random factor in selecting which packets to drop and the "full queue" problem by dropping packets early, before the queue fills.

The RED algorithm uses a weighted average of the total queue length to determine when to drop packets. When a packet arrives at the queue, if the weighted average queue length is less than a minimum threshold value, min_{th}, no drop action will be taken and the packet will simply be enqueued. If the average is greater than min_{th} but less than a maximum threshold, max_{th}, an *early drop* test will be performed as described below. An average queue length in the range between the thresholds indicates some congestion has begun and flows should be notified via packet drops. If the average is greater than the maximum threshold value, a *forced drop* operation will occur. An average queue length in this range indicates persistent congestion and packets must be dropped to avoid a persistently full queue. (The *forced drop* is also used when the queue is full but the average queue length is still below the maximum threshold.) Note that by using a weighted average, RED avoids over-reaction to bursts and instead reacts to longer-term trends. Furthermore, because the thresholds are compared to the weighted average (with a typical weighting factor, w_q, of 1/512), it is possible that no forced drops will take place even when the instantaneous queue length is quite large. For example, Figure 1 illustrates the queue length dynamics in a RED router used in our experiments. For the experiment illustrated in Figure 1, forced drops would occur only in the one short interval near the beginning when the weighted average reaches the maximum threshold.

The *early drop* action in the RED algorithm probabilistically drops the incoming packet when the weighted average queue length is between the min_{th} and max_{th} thresholds. In contrast, the *forced drop* action in the RED algorithm is guaranteed to drop the incoming packet. In the case of early drops, the prob-

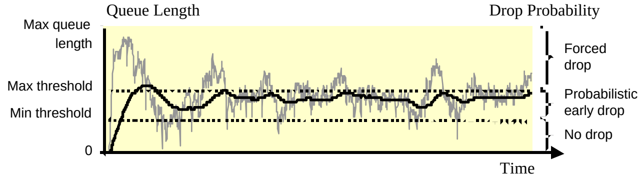

Figure 1: An illustration of the desired behavior of a packet queue at an outbound interface in a router. The gray line indicates instantaneous queue length; the black line indicates the weighted average queue length.

ability that the packet will be dropped is dependent on several other parameters of the algorithm. An initial drop probability $P_b = max_p(avg - min_{th})/(max_{th} - min_{th})$, is computed, where max_p is the maximum drop probability (an additional control parameter) and *avg* is the weighted average queue length. The actual drop probability is a function of the initial probability and a count of the number of packets enqueued since the last packet was dropped: $P_a = P_b/(1 - count \times P_b)$. Note that given a weighted average queue size, the impact of min_{th} is dependent on both max_p and max_{th}. This means that one may find a value for min_{th} that results in good performance, but it may only be in combination with certain values of max_p and max_{th}. In principle, this is the case for all the parameters. The main control parameters for RED are summarized in Table 1.

The design of RED is such that during the drop phases of the algorithm, high bandwidth flows will have a higher number of packets dropped since their packets arrive at a higher rate than lower bandwidth flows (and thus are more likely to be dropped in an early drop action). However, all flows experience the same loss *rate* under RED. By using probabilistic drops, RED maintains a shorter average queue length, avoiding lockout and repeated penalization of the same flows when a burst of packets arrives.

The original RED paper [13] presented analysis and several simulations to show the results of RED usage and develop insights into the effects different RED parameters have on performance. They arrived at suggested guidelines for useful ranges of parameter values and explanations of the considerations that would influence tuning parameters to achieve desired results for particular traffic characteristics. Subsequent analysis by RED's designers and others led to the current guidelines ([15]) that are discussed later in this paper.

One of the earliest experiments with RED was reported in [26] and gives the results of live testing with a RED implementation in a router ahead of a bottleneck DS3 link in a transcontinental network. These tests were conducted with a small number of continuously sending high-bandwidth TCP connections. Total throughput of the TCP connections was the primary measure of performance and delays were not measured. The results showed that, in general, RED achieved better throughput and better link utilization for multiple connections than comparable tail-drop FIFO. RED was also effective in preventing congestion collapse when the TCP windows were configured to exceed the storage capacity of the network. A very important result showed that the interface queue (buffer) size is a critical parameter even with RED and should be 1-2 times the bandwidth-delay product at a bottleneck link.

Table 1: RED control parameters.

qlen	The maximum number of packets that can be enqueued.
min_{th}	Queue length threshold for triggering probabilistic drops.
max_{th}	Queue length threshold for triggering forced drops.
w_q	Weighting factor for the average queue length computation.
max_p	The maximum probability of performing an early drop.

A number of research efforts have focused on possible shortcomings of the algorithms in RED and have proposed modifications and alternatives, among them BLUE [12], SRED (Stabilized RED) [23], Adaptive RED [11], FRED (Flow Random Early Drop) [17], and BRED (Balanced RED) [2]. We do not comment here on the contributions and merits of these proposals except to note any analysis or simulations that examine the behavior and performance of "classic" RED. For example, in [17] simulations are used to demonstrate situations in which RED does not provide protection from non-adaptive flows, and situations in which RED does not promote fair sharing of link bandwidth between TCP connections with long RTT or small windows, and other competing flows.

In [12] there are a suite of results from **ns** simulations of RED with ECN (*explicit congestion notification* [14]) enabled in both routers and end-system TCP implementations. The simulations focused primarily on the effects of the parameter w_q used to smooth measurements of the average queue size. Interestingly, some of these simulations use a large number of sources (1,000-4,000) that generate traffic with Pareto on/off periods and might provide clues to behavior in web-like traffic. Unfortunately, because all the simulations use ECN marking instead of packet drops, and end-to-end delays are not considered, the results are not directly comparable to our work on packet-drop RED. Feng *et al.* presents **ns** simulations of RED with packet drops in situations where a moderate number (32 or 64) of continuously sending TCP connections share a link [11]. Here the maximum drop probability max_p was varied to see its effect on loss rates and average queue length. The results show that the "best" value for max_p is dependent on the number of connections and, for any setting, the drop rate is not significantly different from that of a tail-drop FIFO queue. The argument is also made that the effectiveness of RED decreases as the number of connections sharing the queue increases. This is because a small number of connections actually receive and act on RED-induced congestion indications.

Results reported in [23] for simulations of RED with persistent (continuously sending) TCP connections (ranging from 10-1,000 connections) showed that router queue lengths (measured in the total buffer space consumed) were at or below the minimum threshold for a small number of connections and stabilized around the maximum threshold for a large number of connections. Simulations were also conducted with more "realistic" traffic by using a large number of TCP connections (2,000-3,500) to transfer random size files with a size distribution derived from measurements of Web transfers [6]. Between file transfers, the TCP connections were idle for a "think time" also based on the same data (but with the mean reduced by a factor of 10 to generate a heavier load). The only results reported from simulations with these traffic conditions, however, were for buffer occupancy in the RED router which again demonstrated a tendency to stabilize around the maximum threshold for larger numbers of active flows.

Recent work at INRIA has used analytic models and simulation [20] along with live testing on a commercial RED implementation [19] to quantify the performance effects of RED. The emphasis was on quantifying how RED influences loss rates, patterns of consecutive loss, mean delay, and delay jitter for mixes of "bursty" (TCP) and "smooth" (UDP) traffic, when compared with tail-drop FIFO queue management. The results from analytic models were confirmed with **ns** simulations for a number (up to 300) of continuously-sending TCP connections sharing a bottleneck link with UDP flows operating at 10% of the link capacity. They concluded that TCP "goodput" does not improve significantly with RED and this effect is largely independent of the number of flows. They also observed that the mean queuing delay is lower with RED but has a much larger delay variance. In essence, the RED router behaved as a tail-drop router with a queue length equal to the maximum threshold

Even though the INRIA work considers the effect of both queuing delay and drop rates at routers, it does not integrate these effects with the dynamics of TCP congestion control and retransmission to determine the overall result on end-to-end response times for interactive or web-like traffic. Moreover, the goal in these experiments was to explore how changes in Cisco's WRED configuration parameters could be used to control performance. The measures of performance were throughput, bytes sent, and percentage of UDP drops. There were no measurements of delays or end-to-end response times. Their conclusion was that determining the best combinations of RED parameters is difficult and, overall, RED did not show much better performance than tail-drop FIFO (except with larger queue sizes where RED did show some improvement in performance).

We are aware of only two available reports from network operators that have conducted pilot tests of RED in production – those by Doran at Ebone [7] and Reynolds at QualNet (now Verio) [24]. Doran's measurements using the Cisco implementation indicate that RED was able to sustain near 100% utilization on a 1,920 Kbps customer-access link where tail-drop FIFO could not. Reynolds used the Cisco implementation of WRED on both a DS3 core network link and a DS1 customer-access link. For the heavily congested periods on the core link, it was found that a wide separation of queue thresholds (min_{th} = 60, max_{th} = 500) produced the best tradeoff for link utilization and low drop rates and was somewhat superior to tail-drop FIFO. The default values for drop probability (1/10) and smoothing factor (1/512) were used and their effects not studied. For the customer access DS1 links, (apparently) the default settings were used. These links were congested only during some intervals and some increase in end-to-end latency was observed with RED but the claim was made that "*… the user is not, in my opinion, inconvenienced, and has the benefit of limited packet loss…*" [24].

In summary, while the results from these studies have added important pieces of evidence to the growing corpus of information about RED, important elements are missing. In particular, none of the work we found explicitly considers RED interactions with Web-like traffic where end-to-end response time is the primary measure of performance. Further, many of the results on RED performance are based on "best case" simulations in which a constant number of TCP connections, each sending continuously, share a queue facing a bottleneck link. In the work reported here, we consider the opposite "worst case" in which there is a dynamically changing number of TCP connections with highly variable lifetimes.

3. Experimental Methods

3.1 Experimental Network

For our experiments we constructed a laboratory network that models an enterprise or campus network having a single wide-area link to an upstream Internet service provider (ISP). All traffic using the ISP link is Web traffic where the requesters (browsers) are all located on the enterprise or campus network and all the requests are satisfied by Web servers located somewhere on the Internet beyond the ISP link.

The laboratory network used to emulate this configuration is shown in Figure 2. All systems shown in this figure are Intel architecture machines running FreeBSD 2.2.8. At one edge of this network are machines that run instances of a Web request generator (described below) each of which emulates the browsing behavior of hundreds of human users. At the other edge of the network are another set of machines that run instances of a Web response generator (also described below) that creates traffic in response to the browers' requests. 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 10/100 Mbps Ethernet interfaces configured to run at only 10 Mbps and are attached to a switched VLAN on a Cisco Systems Catalyst 5000. All browser machines are on one VLAN and all server machines are on a separate VLAN.

At the core of this network are two router machines running the ALTQ version 1.2 extensions to FreeBSD. ALTQ extends

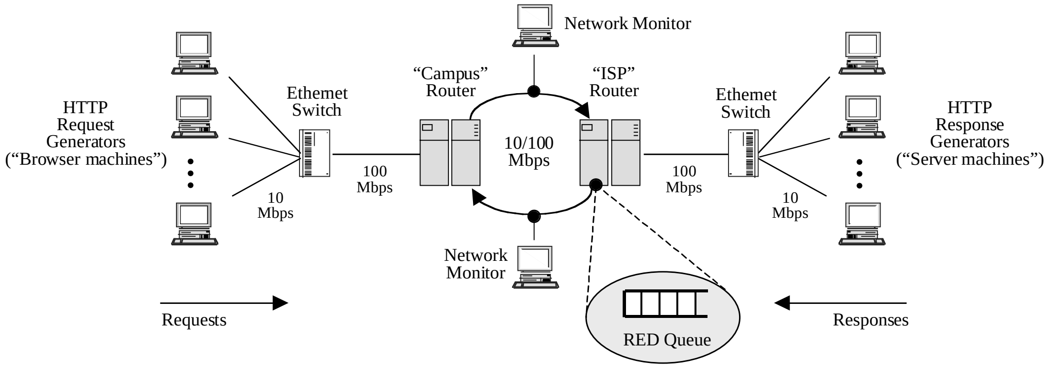

Figure 2: Experimental laboratory network diagram.

the network-interface output queuing discipline to include FIFO, RED, CBQ, and WFQ queue management [16]. These router machines are 300 Mhz Pentium IIs. Each router machine has one 100 Mbps Ethernet interface attached to one of the switched VLANs on the Catalyst 5000. Each router machine also has two additional 10/100 Mbps Ethernet interfaces configured to create two point-to-point Ethernet segments (using two hubs) that connect the routers as shown in Figure 2. Static routes are configured on the routers so that all traffic flowing from the servers to the browsers uses one Ethernet segment and all traffic flowing in the opposite direction uses the other Ethernet segment. This configuration allows us to approximate the full-duplex behavior of the typical wide-area link to an ISP from a customer's network. By configuring the router-to-router Ethernet segments to run at only 10 Mbps, we can make our representation of the ISP link be a potential bottleneck since the aggregate bandwidth available to the machines at each edge of the network is constrained only by the 100 Mbps links from the VLANs to the routers. When the links connecting the routers are configured to run at 100 Mbps, the bottleneck is removed.

Another important factor in modeling this configuration is the effect of end-to-end latency. We use the *dummynet* [8] component of FreeBSD to configure in-bound packet delays on the end systems to emulate different round-trip times between each paring of a browser machine and a server machine. The delays ranged from 7-137 milliseconds and were derived from measurement data obtained at the *NetStat.net* web site [21]. (See Appendix A for the actual delay values used.) The delays were chosen to represent a sample of Internet round-trip times within the continental U.S. A given delay represents the minimum round-trip time experienced by an arbitrary TCP connection between a given pair of client and server machines in our experiments (assuming no delays in the two routers). As explained below, the distribution of TCP connections over pairs of machines should be approximately uniform and, thus, we can calculate the mean minimal round-trip time for all TCP connections sharing the network as approximately 79 milliseconds. The default TCP window size in FreeBSD of 16K bytes was used on all the end systems. (For other characteristics of the TCP implementation, see Appendix A.)

The instrumentation used to collect network data during runs of the experiments consists of two monitoring programs. One monitor is on the router interface where we are examining the effects of queue algorithms. It calculates a mean and variance of the queue size sampled every 3 milliseconds. The maximum and minimum queue size seen in any sample is also collected. These statistics are logged every 100 milliseconds along with more general information about the number of transmitted and dropped packets. The second monitoring program runs on a separate machine connected to the hubs forming the links between the routers (see Figure 2). Using a modified version of the *tcpdump* utility, the machine collects the TCP/IP headers in each frame traversing the links and processes these to produce a log of link throughput over each specified time interval (typically one second). End-to-end performance measures such as response times are measured on the end-systems as described below.

3.2 Web-like Traffic Generation

The traffic that drives the experiments described here is based on the model of web browsing developed by Mah [18]. Mah's model is an application-level description of the critical elements that characterize how HTTP 1.0 [22] protocols are used. It is based on empirical data and is intended for use in generating synthetic Web workloads. The data were extracted from more than 230 hours of traces collected on the UC-Berkeley campus in late 1995 and include over 1.6 million HTTP protocol packets. These data were used to compute empirical distributions describing elements necessary to generate synthetic HTTP workloads. The elements of the HTTP model are:

- HTTP request length in bytes,
- HTTP reply length in bytes,
- Number of embedded (file) references per page,
- Time between retrieval of two successive pages (user "think" time), and
- Number of consecutive pages requested from a server.

The empirical distributions for all these elements are used in synthetic-traffic generator programs we wrote. The elements that have the most pronounced effects on generated traffic are the size of server responses, the number of requests necessary to download a page (including all embedded references), and the user "think" time between successive page requests. (See Appendix A for a more detailed discussion of the Mah data.) We used the Mah model to write Web-traffic generating programs using the normal *socket* system calls provided in FreeBSD. Most of the behavioral elements of Web browsing are emulated in the client-side request-generating program. Its primary parameter is the number of browsing users (typically several hundred) the program is to represent. For each user, 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 separate TCP connection, as implied by the HTTP 1.0 protocol, is made to the server-side portion of the program for the primary page and each embedded reference (the distribution of embedded references per page is used to generate a random value). Another parameter of the program is the number of concurrent TCP connections allowed on behalf of each browsing user to make embedded requests within a page (this parameter is used to mimic the behavior of Netscape and Internet Explorer).

For each request, a message of random size (sampled from the request size distribution) is sent to the server program. This message contains a value that represents the number of bytes the server is to return as a response (a random sample from the distribution of response sizes). The server sends this number of bytes back to the browser and closes the TCP connection. 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 roughly models the behavior of a Web server or proxy having a large main-memory cache with a hit-ratio near 1.0). For each request/response pair, the browser program logs its response time. Response time is defined as the elapsed time in milliseconds between the time of the socket *connect()* operation and the time the response is completed and the connection is closed. 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 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 server to handle subsequent requests is selected randomly and uniformly from the set of active servers. The number of emulated users is constant throughout the execution of each experiment.

The HTTP 1.0 protocol implies the use of a new TCP connection for each request/response pair. This protocol is gradually being replaced by the more efficient HTTP 1.1 protocol which allows multiple and pipelined requests to reuse TCP connections [22]. While some data have been reported (*e.g.* [10]) suggesting that as many as 30% of HTTP requests now use the HTTP 1.1 protocol, we have been unable to find data or models sufficient for building a synthetic workload generator for HTTP 1.1. For these reasons we generate only HTTP 1.0 traffic in our experiments. We note, however, that the older HTTP 1.0 protocols are expected to represent a very significant portion of Web traffic for some time because of difficulties with migrating the installed base of browsers. Furthermore, our focus on HTTP 1.0 serves as a worst-case analysis of RED performance.

3.3 Experiment Calibrations

There are two 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 when the links connecting the two routers are limited to 10 Mbps, and (2) the offered load on the network can be predictably controlled using the number of emulated browsing users as a parameter to the traffic generator. To perform these calibrations, we first configured the two segments connecting the routers to eliminate congestion by running at 100 Mbps.

The first calibration performed was to verify that the traffic generator programs did not have any resource constraints that limited their ability to emulate hundreds of users. These programs were implemented using efficient programming techniques for managing large numbers of socket connections (based in part on Banga and Druschel's scalable methods for generating HTTP requests [3]). For this calibration we first selected the slowest machine in our network (a 66 Mhz 486) to run the browser program. We ran one instance of the server-side program on each of the server machines and configured the browser program to select uniformly from all servers for each new sequence of page requests. The number of browsing users was varied from 500 to 1,400 and the bandwidth used on the 10 Mbps interface to the browser machine is plotted in Figure 3 as a function of the number of simulated browsing users. These results show that over this range of users, there is a linear increase in generated traffic and the traffic is significantly less than the capacity of the host's 10 Mbps interface. We repeated this experiment with a 200 Mhz Pentium Pro with the results also shown in Figure 3 for further confirmation that CPU and interface speeds of the end system are not resource constraints. Thus if traffic generation machines are limited to simulating no more than 1,400 users each, we can be confident that the number of users simulated in an experiment is accurate and reproducible.

A second concern is that a single program can not faithfully simulate hundreds of browsers because by default, a single FreeBSD process can use at most 64 sockets simultaneously. However, because user think times are much longer than the times required to request pages, most of the emulated users are idle at any time. We explicitly performed experiments to demonstrate that the 64 socket descriptors limitation was never encountered in practice. With a similar experiment we also verified that even the slowest server machine could handle a maximum number of expected requests without reaching a resource limitation.

For the next calibration, we ran an instance of the browser program on each of the browser machines and again uniformly distributed requests across all server machines. Each browser was configured to emulate the same number of users with the total users varied from 700 to 5,075. Figure 4 plots aggregate traffic on the path carrying response traffic from the servers as a function of emulated browsers (users). Again the load is a linear function of browsers indicating there are no fundamental resource limitations in the system and generated loads can easily exceed the capacity of a 10 Mbps link.

With these data we can determine the number of emulated browsers that would generate a specific offered load in this configuration if there were no bottleneck link present. This capability is used in subsequent experiments to control the offered loads on the network, including loads that nominally exceed the capacity of a 10 Mbps link. For example, if we want to generate an offered load equal to the capacity of a 10 Mbps link, we use Figure 4 to determine that we need to emulate approximately 3,400 browsing users; for a load of 110% (11 Mbps) we need to emulate 3,750 users.

A motivation for choosing Web-like traffic to drive these experiments was the assumption that properly generated traffic would exhibit highly variable and bursty demands on the network. To illustrate that this is indeed realized with our experimental setup, we have plotted the results from one of the calibration experiments (3,500 browsers) in Figures 5 and 6. These plots show the number of requests initiated during each one second interval (each request requires a new TCP connection) and the number of bytes requested (not necessarily received) in each one second interval. Clearly these show the highly bursty nature of the traffic actually generated.

3.4 Experimental Procedures

Each experiment was run using the following procedure. After initializing and configuring all router and end-system parameters, the server-side processes were started followed by the browser processes. Each browser emulated an equal number of users chosen, as described above, to place a nominal offered load on an unconstrained (100 Mbps) network. The offered loads used in the experiments were chosen to represent 50, 70, 80, 90, 98, or 110 percent of the capacity of a 10 Mbps link connecting the two router machines. Loads exceeding 110% were tried; it turned out, however, that the extreme duration of the connections when using a congested link caused the traffic generators to occasionally use all available sockets and fail to generate the desired level of traffic. Because the measured response times at a load of 120% had deteriorated well beyond levels that most users would tolerate, we decided to not consider loads beyond 110% on the congested link.

Each experiment was run for 90 minutes but data collected during the first 20 minutes was discarded to eliminate startup and stabilization effects. These effects are illustrated in Figure 7 which shows a plot of mean response times for requests during each one second interval in a typical experiment. Figure 8 gives a plot of the cumulative distribution of response times at a load from 3,500 browsers in an unconstrained network. Note that about 90% of the requests complete in 500 milliseconds or less. Figure 8 represents the best-case performance for HTTP request/response pairs and will be used as a basis for comparison with experiments on the constrained (congested)

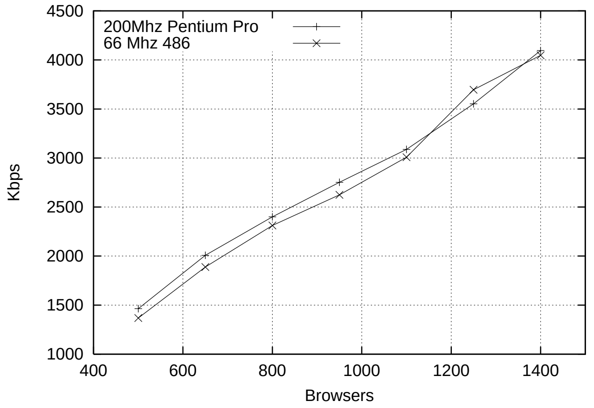

Figure 3: Offered load as a function of the number of simulated users on one machine.

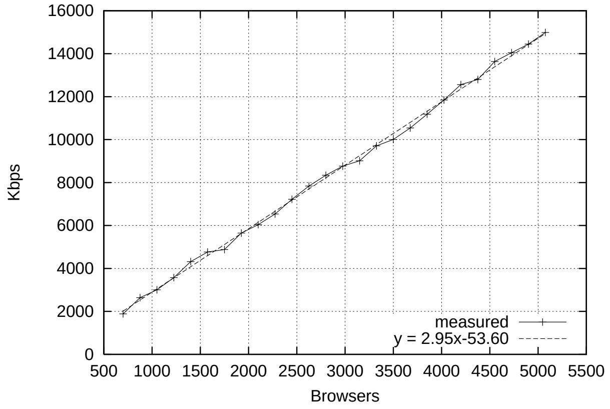

Figure 4: Offered load as a function of the number of simulated users on 7 machines.

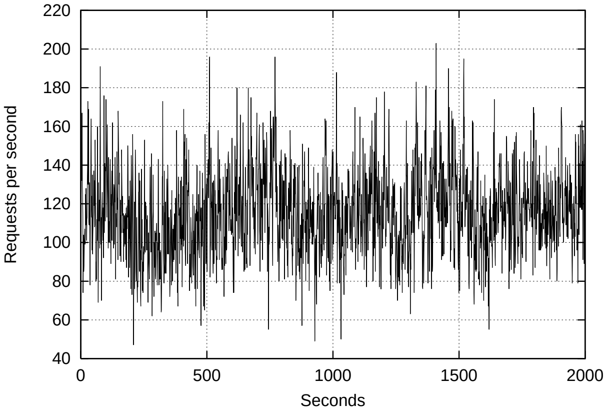

Figure 5: Requests per second from 3,500 users.

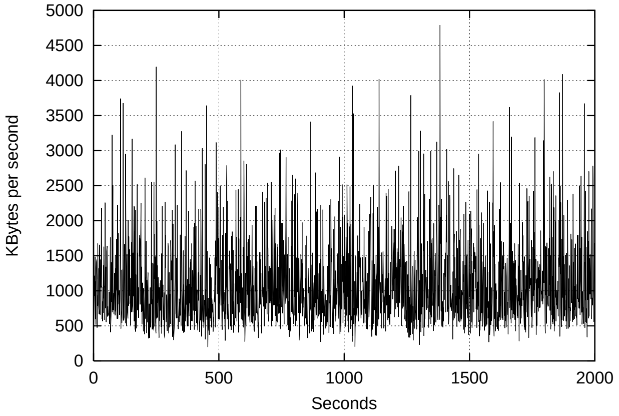

Figure 6: Bytes requested per second from 3,500 users.

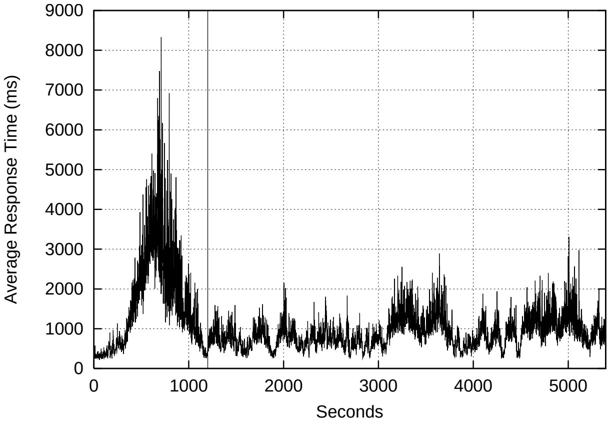

Figure 7: Average response time per second during an experiment. The plot includes the initial 20 minutes, where the traffic generators are started and stabilize.

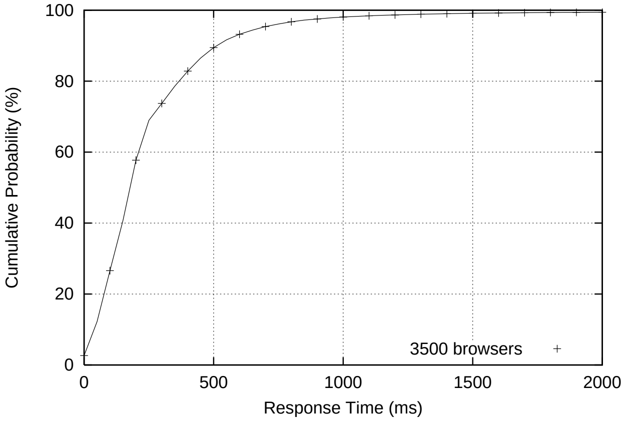

Figure 8: Cumulative response time distribution for 3,500 users on the unconstrained (100 Mbps) network.

network link. Table 2 shows the number of requests generated during a 70 minute interval for each of the loads in typical runs on the unconstrained network.

Because responses are much larger than requests, the load on the link between routers that carries traffic from the servers to the browsers (the link from the "ISP" router to the "campus" router in Figure 2) will be much greater than that on the link carrying traffic in the opposite direction. Consequently, only the effects of different queue management algorithms on the IP output queue for this link interface are reported here. The IP output queues for the link interfaces on all other machines in the network were tail-drop FIFO queues with the FreeBSD default queue size of 50 elements. Data collected on these interfaces using the *netstat* function showed no dropped packets.

The key indicators of performance we use in reporting our results are the end-to-end response times for each request/response pair. We report several measures of response times including the median, the percent of requests complet-

Table 2: Typical numbers of requests in a 70 minute interval.

Load %	Requests	Load %	Requests
50	240,379	90	425,293
70	329,638	98	461,837
80	375,673	110	521,561

ing in intervals of 0-1, 1-2, 2-3, and greater than 3 seconds, and plots of the cumulative distributions of response times (usually showing only times less than or equal to 2 seconds). We also measured the percent of IP datagrams dropped at the link queue, the mean queue size, and the link throughput actually achieved on the bottleneck link.

4. FIFO Results

To establish a baseline for evaluating the effects of using RED on interface queues for links carrying only Web traffic, we first examined the effects of FIFO queues with tail-drop behavior in our experimental network. For these experiments we created a bottleneck between the two routers by configuring the two segments connecting the router machines to run at 10 Mbps using 10 Mbps hubs (see Figure 2). The critical parameter for a FIFO queue is the size of the buffer space allocated to hold the queue elements. Guidelines (or “rules of thumb”) for determining the “best” queue size have been widely debated in various venues including the IRTF *end2end-interest* mailing list [9]. The 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 for the interface using the queue and delay is the mean round-trip time for all connections sharing the link – a value that is, in general, very difficult to determine. For our experimental network, the mean minimum round-trip time can be computed as 79 milliseconds and the 10 Mbps link has a bandwidth-delay product of approximately 96K bytes (see Appendix A). 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. We measured the mean IP datagram size in our generated Web response traffic to be just over 1K bytes so the FIFO queue should have approximately 190-380 queue elements to fall within the guidelines.

We ran a number of experiments with a FIFO queue on the bottleneck link varying the offered load and queue size. Figure 9 shows the cumulative response time distributions for different FIFO queue sizes at loads of 80%, 90%, 98%, and 110%. At a load of 80%, there is little effect from increasing the queue size from 30 to 240 elements. At 90% load we begin to see queue size having more significant effects on response times and observe that a queue size of 120 elements is a reasonable choice for this loading. The effect that queue size has on response times depends on the size of the HTTP response data as is shown in the plots for 98% load. Increasing the queue size from 30 to 120 has a slightly negative effect on relatively short responses that could complete in a few hundred milliseconds by increasing the amount of time each packet spends in the queue. For a 10 Mbps Ethernet link and an average frame size around 1 KB, approximately 1,000 packets can be for-

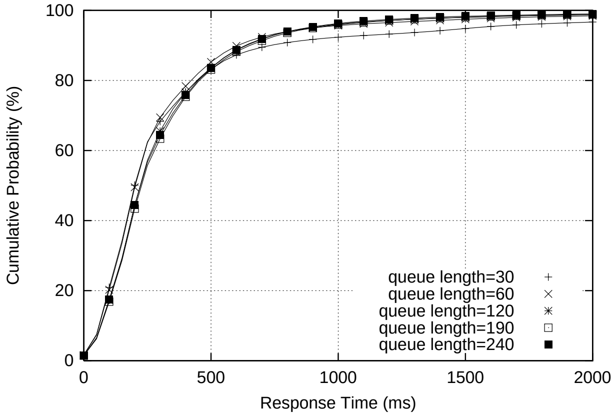

Figure 9a: FIFO performance at 80% load.

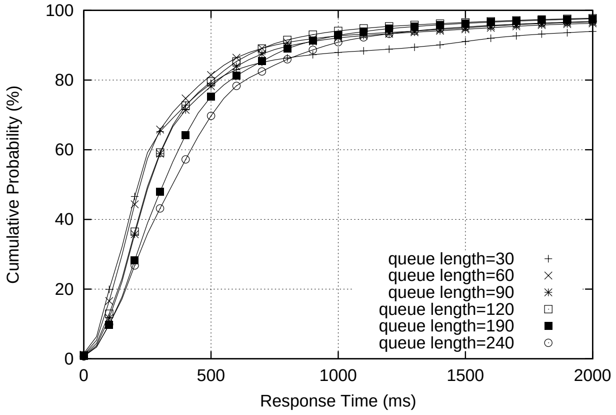

Figure 9b: FIFO performance at 90% load.

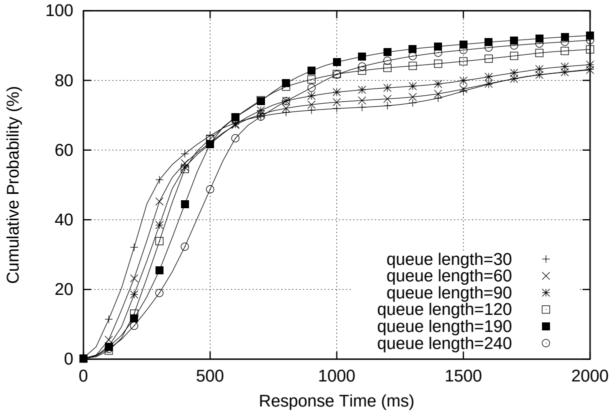

Figure 9c: FIFO performance at 98% load.

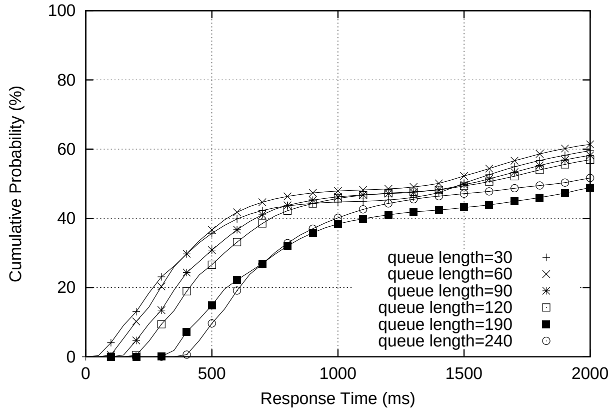

Figure 9d: FIFO performance at 110% load.

warded per second. Thus a packet arriving at the queue already containing 100 packets has to wait approximately 100 milliseconds on the router. Such a delay is significant for requests with short responses that may otherwise complete within 200-350 milliseconds. On the other hand, increasing the queue size from 30 to 120 reduces response times significantly for long requests. Even though the time spent in the queue by each packet is longer, the reduced rate of drops means that longer responses are less likely to encounter retransmission timeouts (which are often longer than queuing delays by a factor of 5-10 times). At queue sizes of 190 or 240 the increase in response times for short requests appears to offset any improvement gained for longer requests from reduced drops.

Our results indicate that, overall, a FIFO queue size of 120 elements (about 1.25 times the bandwidth-delay product) to 190 elements (2 times bandwidth-delay) is a reasonable choice for loads up to the link capacity. For offered loads that only slightly exceed the link capacity (*e.g.*, 110%), we observe that queue sizes beyond 120 only exacerbate an already bad situation. Additional measures of performance in these experiments, including link utilization and drop rates, are given in Appendix B. These results confirm that our selection of queue sizes of 120-190 represent reasonable tradeoffs for response times without significant loss of link utilization or high drop rates.

These experiments illustrate (as queuing theory predicts) the dramatic effect that offered loads near or slightly beyond the link capacity have on response times. Figure 10 shows the cumulative distribution of response times for these loads with a FIFO queue of 120 elements. Clearly, response times degrade sharply when the offered load approaches or exceeds link capacity. If an ISP has links that experience utilization above 90% over intervals greater than a few minutes, response time for Web users are seriously impacted. A second important observation is that at loads below 80% there is no significant change in response times as a function of load.

5. RED Results

The goal for our experiments with RED was to determine parameter settings that provide good performance for Web-traffic. We also examined the tradeoffs among the different parameters in tuning for performance. The RED queuing mechanism has five different parameters for adjusting the algorithm's behavior. An exhaustive search for the best parameter values is impossible because of the number of possible combinations of values. Our approach for the RED experiments was to design an initial set of experiments that could give a broad approximation of parameter values that result in good HTTP performance. We then examine the effects of varying each parameter individually using this initial determination as a baseline.

From our experiments with FIFO it is clear that there is a complex tradeoff between response times for short responses that can be completed in a few hundred milliseconds (best with a short queue) and response times for longer responses (best with longer queues and lower drop rates). The original Floyd and Jacobson paper [13] suggests guidelines for tuning parameters that have been revised based on subsequent experience and analysis (see [15] for the current guidelines). These guidelines suggest that the most fundamental effects are determined by the min_{th} and w_q parameters which control tradeoffs between average queue size and sensitivity to the duration of periods of congestion. For our initial experiments we decided to eliminate the size of the physical queue as a factor and set the number of queue elements to 480, more than double the largest average queue size seen in the FIFO experiments. In these experiments we varied min_{th} beginning with the guideline value of 5 and ranging up to 120. We fixed max_p at 0.10, w_q at 0.002 (actually 1/512), and max_{th} at 3 times min_{th} as suggested in the current guidelines.

Each of the parameter settings was tried at six different offered loads: 50%, 70%, 80%, 90%, 98%, and 110%. At 50% load the number of dropped packets was between 0.00% and 0.01% of the total number of packets transmitted. This means that at loads of 50% and below, there is limited room for increasing the performance of the router queuing mechanism. Post processing of the logs shows that the queue size never reaches the maximum value of 480 even at a load of 110%, though it is possible in a worst-case scenario. As expected, the performance changes significantly as the load is increased from 50% to 110%. Figure 11 illustrates typical results from these experiments by showing the effect of varying loads on response time distributions with (min_{th}, max_{th}) set to (30, 90).

It is encouraging to see that performance degradation only occurs at loads greater then 70%, especially when combined with the fact that the drop rates at 50% load never exceeds 0.01% of the packets received at the router. This indicates that parameter tuning will have limited effect until loads reach levels of 70-80% of link capacity. When loads exceed 70%, the performance decreases monotonically as the load in-

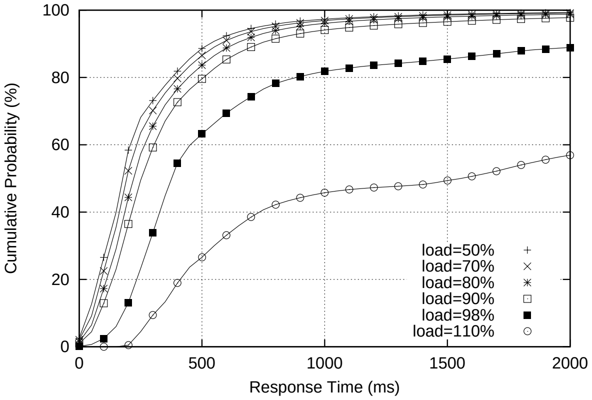

Figure 10: FIFO performance for different loads with a queue length of 120 elements.

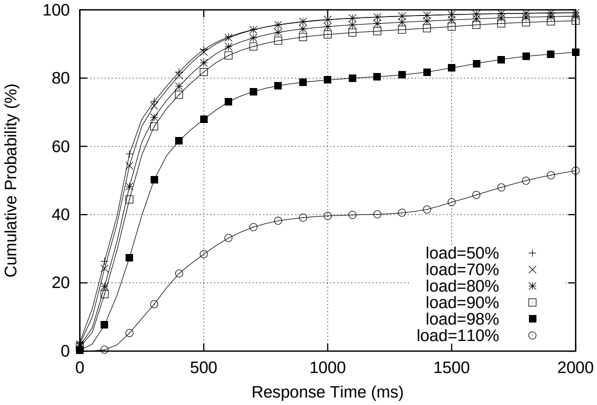

Figure 11: The performance of RED at different loads. w_q=1/512, max_p=1/10, min_{th}=30, max_{th}=90, $qlen$=480.

creases. The most significant performance decrease occurs at load levels of 90-110%. These are the most interesting targets for optimization, since this is where there is significant performance to gain.

We start by exploring possible choices for min_{th} and max_{th}. Figure 12 shows the response time distributions for the 90% and 98% offered loads, respectively. These results clearly show that a naive application of the guidelines in [15] with a min_{th} of 5 would result in poor performance for Web-dominated traffic. The best overall response-time performance is obtained with values for (min_{th}, max_{th}) of (30, 90) or (60, 180). We see, as in the case of FIFO, that there is a tradeoff between better response times for short responses at (30, 90) and improving response times for longer ones at (60, 180), especially at the 98% load. Although the differences are not great, we prefer (30, 90) on the grounds that about 70% of the requests experience somewhat better response times than with (60, 180). (One could also argue that (60, 180) is best because it improves the most noticeable delays.) The complete results, including link utilization and drop rates for loads of 90%, 98%, and 110% are summarized in Appendix B. These indicate a slight drop in link utilization for the (30, 90) setting over the (60, 180) setting. Like the FIFO results, response times at loads of 110% are quite bad and are not improved by changing the RED settings for (min_{th}, max_{th}).

We next consider varying the ratio between min_{th} and max_{th} by holding one constant and varying the other. To see the effect of min_{th}, we first fixed max_{th} at 90 and varied min_{th}. We then held min_{th} constant at 30 and varied max_{th}. We fixed max_p at 0.10, w_q at 0.002 (actually 1/512), and *qlen* at 480 as in the previous experiments. Figure 13 illustrates the effect from varying min_{th} on the response time distributions for the 90% load. The results obtained by varying max_{th} are similar. The results from these experiments, in general, show only marginal changes in response times (or link utilization) and confirmed the notion that the best balance of response times for all sizes of responses with the loads considered here are achieved with (min_{th}, max_{th}) = (30, 90). The complete results from these experiments are again given in Appendix B.

Experiments testing the impact of changing w_q and max_p were combined because of the close relationship between the two parameters. The values used for w_q were: 1/512, 1/256, and 1/128. (The implementation of RED requires the denominator to be a power of 2.) Decreasing w_q to 1/1024 was tried, but we found it to be an unrealistic setting that causes reaction to congestion to be quite slow. The values of max_p used were 0.05, 0.10, and 0.25. The remaining parameters were fixed at min_{th} = 30, max_{th} = 90, and *qlen* = 480. All the different settings were tested at loads of 90, 98, and 110%.

These experiments showed that at all load levels the setting of max_p to 0.25 has a negative impact on performance, because too many packets are dropped. Figure 14 shows the results from the experiments at 90% load (the results at 98% are similar). At 90% and at 98% load, the difference between the set-

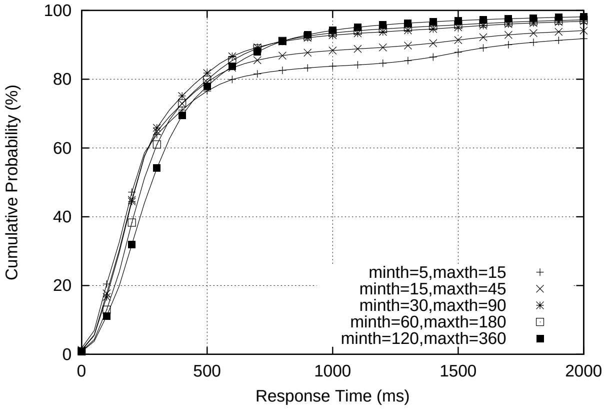

Figure 12a: Response time CDF for offered load at 90% of link capacity (w_q=1/512, max_p=1/10, *qlen*=480).

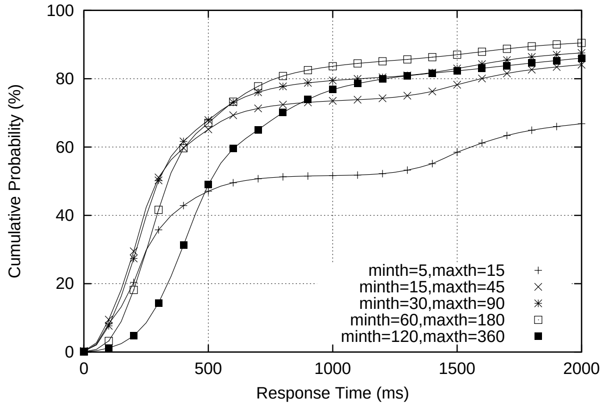

Figure 12b: Response time CDF for offered load at 98% of link capacity (w_q=1/512, max_p=1/10, *qlen*=480).

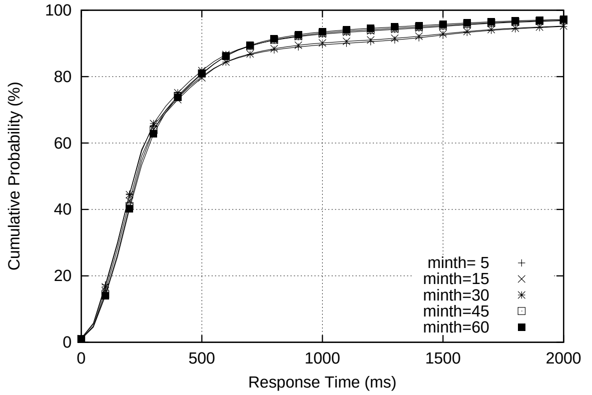

Figure 13: The effect of changing min_{th}. Load = 90% and max_{th} = 90, w_q = 1/512, max_p = 1/10, *qlen* = 480.

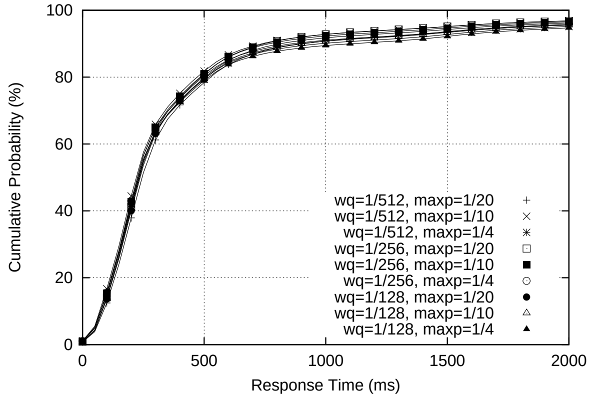

Figure 14: Results for different values of w_q and max_p. Load = 90%, and *qlen* = 480, min_{th} = 30, max_{th} = 90.

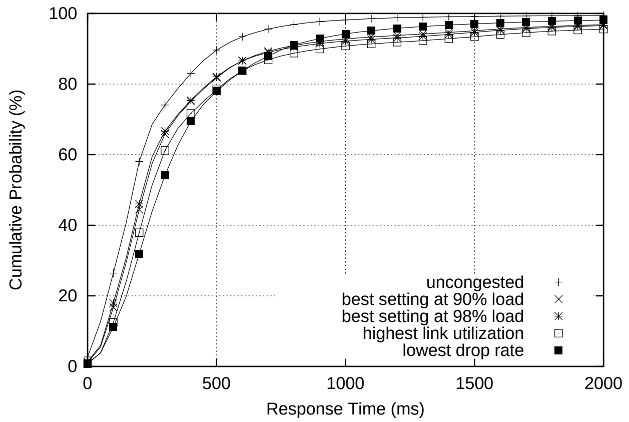

Figure 15a: “Good” RED parameter settings at 90% load.

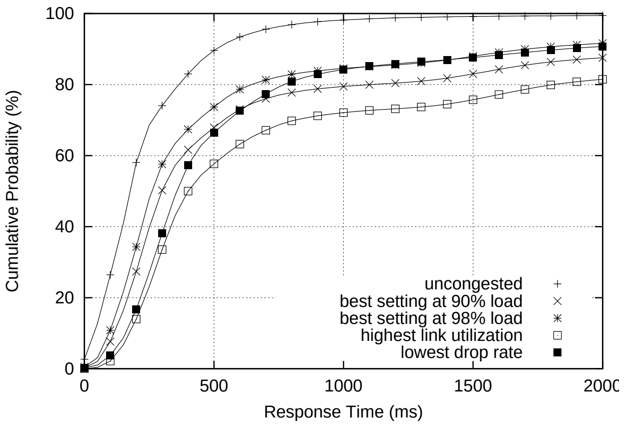

Figure 15b: “Good” RED parameters settings at 98% load.

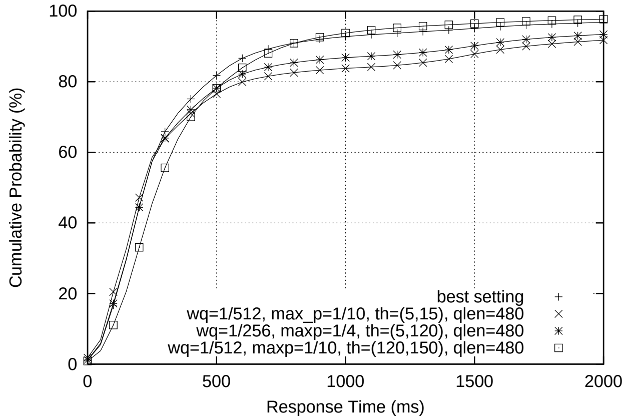

Figure 16a: “Bad” RED parameters settings at 90% load.

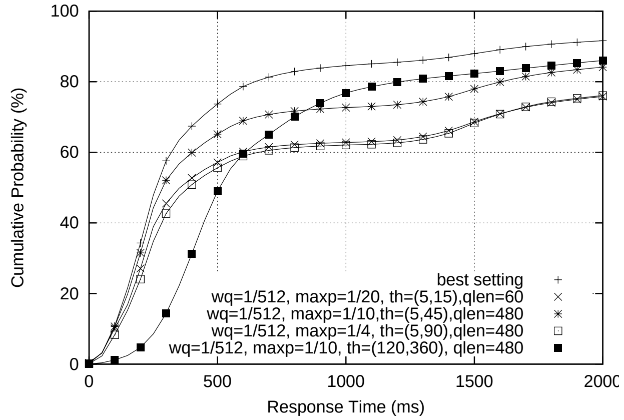

Figure 16b: “Bad” RED parameters settings at 98% load.

tings occurs beyond the knee (above the 75th percentile) of the CDF, meaning that changes of w_q and max_p mainly impact the longer flows. Overall, however, we conclude that there is no strong evidence to indicate using values other than the suggested w_q = 1/512 and max_p = 0.10.

Finally, we consider the effect of having a limit on the queue size such that there are occasionally forced drops because the instantaneous queue exceeds the buffer space. Table 3 gives experimental results with our recommended values of RED parameters for actual queue sizes of 480, 160, and 120 elements. These results are very similar to the FIFO results – the 120 element queue (1.25 times bandwidth-delay) is a reasonable choice at 90% and 110% loads while a longer queue of 2-3 times bandwidth-delay might provide some advantage at loads just below link saturation.

Our conclusion is that, except for min_{th} which should be set to larger values to accommodate the highly bursty character of Web traffic, the guidelines for RED parameter settings and for configuring interface buffer sizes (for both FIFO and RED) also hold for the Web-like traffic used in our experiments. We also conclude that attempting to tune RED parameters outside these guidelines is unlikely to yield significant benefits.

To illustrate the latter point, we examined the entire suite of experiments conducted for the 90% and 98% loads (including some trial experiments with parameter values outside the ranges reported above) to find the combination of settings that gave the best results on three performance measures: “best” response times (a subjective choice because of the trade-off between improving response times for short *v.* long responses), best link utilization, and lowest drop rate. These settings are shown in Table 4 and the response times shown in Figure 15. For 90% load, there are relatively small differences between tuning for highest link utilization or lowest drop rates and tuning for response times. At 98% loads, tuning for highest link utilization has potentially serious effects on increasing response times. Note that the “best” overall response times are obtained for the 98% load (only) with parameters that are quite different from our generally recommended settings. (In Figure 15, the “uncongested” plots refer to the performance on the unconstrained 100 Mbps network.)

Table 3: RED performance with recommended parameters and queue lengths.

Load %	Queue Length	KB/s	% drop	Mean queue	Median resp.(ms)	% ≤ 1 sec	1<%≤ 2 sec	2<%≤ 3 sec	% > 3 sec
90	480	1079	0.8	20.2	266	92.5	4.3	2.0	1.3
90	160	1093	1.1	22.2	278	91.2	4.7	2.4	1.7
90	120	1066	0.7	18.8	266	93.0	4.1	1.7	1.2
98	480	1164	4.1	39.4	345	79.2	8.2	6.3	6.3
98	160	1175	5.9	46.3	397	72.4	9.7	8.2	9.7
98	120	1171	5.5	44.3	377	74.2	9.2	7.7	8.9
110	480	1187	19.7	76.0	1846	39.4	12.9	12.1	35.5
110	160	1188	19.5	76.6	1864	39.1	13.0	12.2	35.7
110	120	1188	18.9	77.0	1840	39.3	13.2	12.5	34.9

Moreover, there is a significant down-side potential for choosing "bad" parameter settings, especially at near-saturation loads. We again searched the entire set of experiments for the 90% and 98% loads looking for combinations of RED parameters that produced response times that (subjectively) represented poor choices (*i.e.*, choices that increased response times significantly for larger numbers of either short or long responses). Figure 16 shows these results. Clearly some parameter settings produce results that are considerably less desirable than our recommended ones.

Table 4: Empirically determined "best" RED parameter values.

Load	min_{th}, max_{th}	w_q	max_p	Notes
90	30,90	1/512	1/10	best overall response
90	30,90	1/512	1/20	highest link utilization
90	120,360	1/512	1/10	lowest drop rate
98	5,90	1/128	1/20	best overall response
98	30,180	1/512	1/10	highest link utilization
98	90,150	1/512	1/10	lowest drop rate

6. Analysis of RED Response Times

While a detailed analysis of the causes of the response time distributions observed under RED remains the subject of further study, we are able to report the results of a preliminary analysis. We repeated two of the RED experiments reported in Figure 12b using more elaborate instrumentation to provide additional data for each HTTP request/response pair. In addition to end-to-end response times, this instrumentation allowed us to determine a detailed breakdown of the number and types of retransmission events occurring in the TCP connections. We repeated RED experiments for an offered load of 98% with two sets of RED parameter values that produced clearly different response time results; (min_{th}, max_{th}) = (5, 15) and (60, 180). In both cases the remaining RED parameters were fixed at max_p = 1/10, w_q = 1/512, and $qlen$ = 480.

Table 5 gives the percentage of TCP connections that experienced no retransmissions, those that experienced one or more retransmissions of a SYN segment, one or more retransmissions of a FIN segment, one or more retransmissions of a data segment, and those with any combination of SYN, FIN, and data segment retransmissions. In this analysis we consider only retransmissions on the congested path from the server to the client (there were no observed packet drops on the reverse path).

Table 5: Summary retransmission statistics for experiments with more detailed instrumentation.

Class of retransmission event	% of all TCP connections (min_{th}, max_{th}) = (5,15)	 (60,180)
No retransmissions	56.1	87.1
1 or more retransmissions	43.9	12.9
1 or more SYN segments	7.4	2.0
1 or more FIN segments	6.0	2.0
1 or more data segments	25.5	8.5
Combined SYN/FIN/data	5.0	0.4
Total TCP connections	439,979	460,022
Total segments lost	12.4%	2.4%

Figures 17 and 18 give the cumulative distributions of response times for those connections that experienced no retransmissions and for the connections that experienced retransmission events of the types described above. Also shown is the cumulative distribution of response times for all connections. In Figure 17 we observe that the response times for about 50% of the connections with FIN or data retransmissions are shifted relative to those with no retransmissions by an amount corresponding to typical retransmission timeouts in our experiments (approximately 1.5 seconds). The response times for connections with SYN retransmissions are shifted even more because of the longer timeouts on TCP connection establishment. Connections with one or more data retransmissions or with combinations of retransmission types have heavier distribution tails (longer response times) because of the cumulative effects of multiple retransmissions.

Comparing the two figures we observe that response times for those connections having retransmissions are longer in Figure 18 by a factor somewhat greater than the additional mean queueing delay for this case (about 55 milliseconds). Our preliminary analysis indicates that changes in response times because of retransmissions are a complex combination of factors that influence the retransmission delays. These include the mean queueing delay (which influences the estimated RTT), the deviations in RTT caused by increased variance in queueing delays (which are magnified by a factor of 4 in the TCP algo-

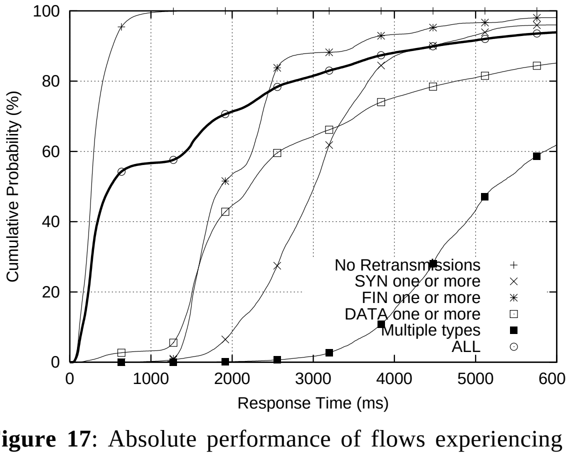

Figure 17: Absolute performance of flows experiencing retransmissions ((min_{th}, max_{th}) = (5, 15)).

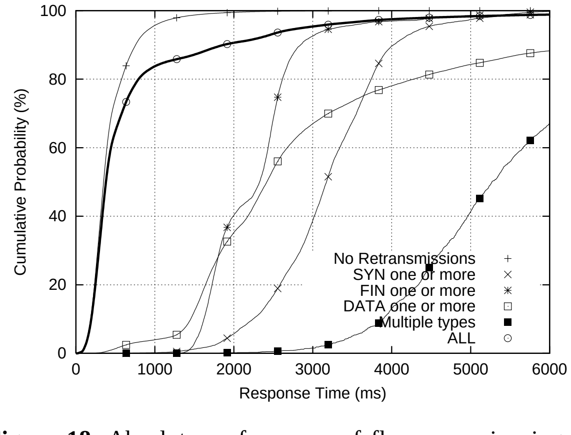

Figure 18: Absolute performance of flows experiencing retransmissions ((min_{th}, max_{th}) = (60, 180)).

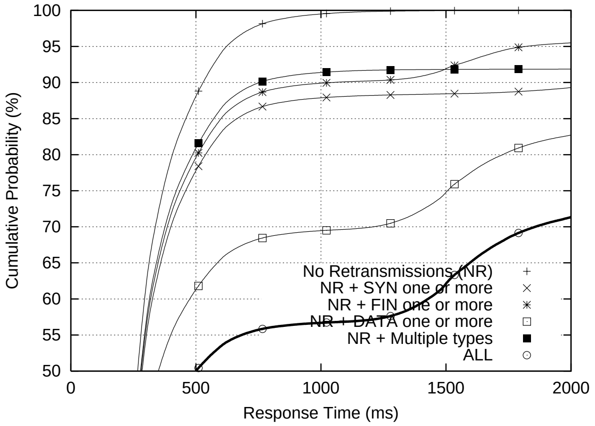

Figure 19: Relative contribution of flows experiencing retransmissions to total distribution ((min_{th}, max_{th}) = (5,15)).

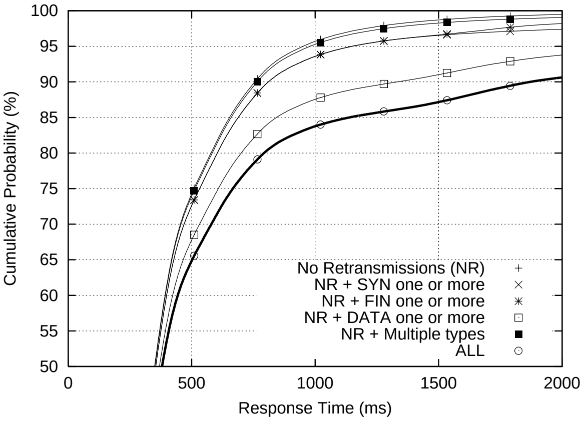

Figure 20: Relative contribution of flows experiencing retransmissions to total distribution ((min_{th}, max_{th}) = (60,180)).

rithm for computing the timeout), the timer granularity (500 milliseconds), and the minimum timeout value (1 second). (For a more comprehensive analysis of these and other factors affecting TCP retransmissions see [1].)

The relative contributions of each class of retransmission to the overall response time distribution is shown in Figures 19 and 20. They show the cumulative distributions of connections with the following characteristics: those with no retransmissions, those with either no retransmissions or only SYN retransmissions, those with either no retransmissions or only FIN retransmissions, and those with either no retransmissions or only data segment retransmissions. To magnify the relative contributions of each class we only show the portion of the distribution beyond the 50th percentile. Contrary to our expectations, retransmissions of lost SYNs (even when most of the TCP connections transfer relatively few bytes) is far from being the dominant factor leading to increased response times. It is, in fact, data segment retransmissions that have the greatest cumulative effect.

Another view of these dynamics is shown in Figure 21 that gives a scatter plot of response times versus server reply sizes. There is one dot in this plot for each of the approximately 400,000 connections with reply sizes less than 16,000 bytes in the experiment illustrated in Figures 17 and 19 ((min_{th}, max_{th}) = (5, 15)). Connections experiencing one or more retransmissions are marked with dark black dots while those with no retransmissions are marked with gray dots. Several features of this plot are striking:

- The large influence of retransmissions on response times for short responses (*e.g.*, the large number of replies of size less then 4K bytes that take 5 seconds to complete),
- The clear regions of response times divided between connections with and without retransmissions,
- The distinct bands of response times at intervals roughly proportional to the granularity of the TCP retransmission timer,
- The sharp step increase in response times with no retransmissions for those responses with lengths greater than 2,880 bytes (corresponding to the initial TCP congestion window), and
- The relatively few connections with retransmissions that avoid a timeout, *e.g.*, with fast retransmission (indicated by black dots in the region dominated by connections with no retransmissions).

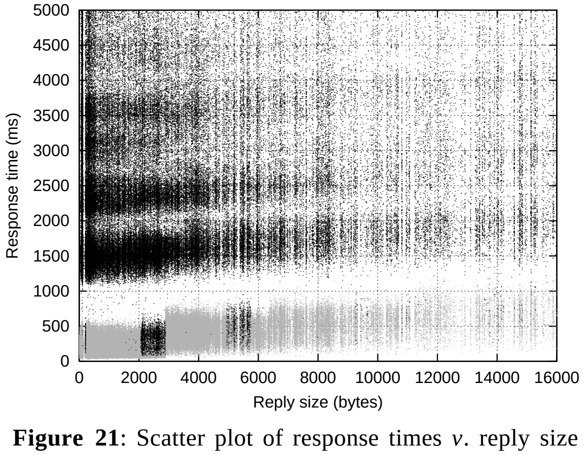

Figure 21: Scatter plot of response times *v*. reply size under RED for (min_{th}, max_{th}) = (5,15).

This brief analysis has re-enforced our view that understanding the effects of RED and FIFO queue management on end-to-end response times for HTTP traffic is a complex issue. It involves many trade-offs and parameters including not only parameters set on routers but also those controlled at the end-systems (*e.g.*, TCP retransmission parameters).

7. Comparing FIFO and RED

Figure 22 shows the response time distributions for RED and FIFO with the parameters selected as a result of our experiments at offered loads of 90%, 98%, and 110%, respectively. Also included for reference are the response time distributions at these loads from the calibrations on the unconstrained network. The only case in which there is a distinct advantage from using RED is at the 98% load where response times for shorter responses (80% of requests) are improved with carefully tuned RED parameters. Note that in Figure 22a we include the response time distribution for both (min_{th}, max_{th}) = (30, 90) and (60, 180). Recall that these settings gave nearly identical performance at 90% load and we selected (min_{th}, max_{th}) = (30, 90) as the best setting at this level for largely subjective reasons.

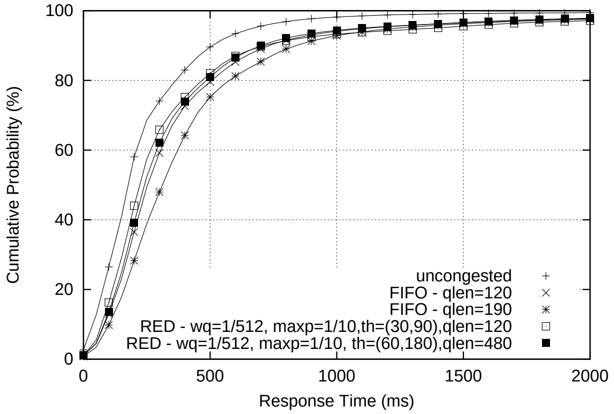

Figure 22a: FIFO and RED at 90% load.

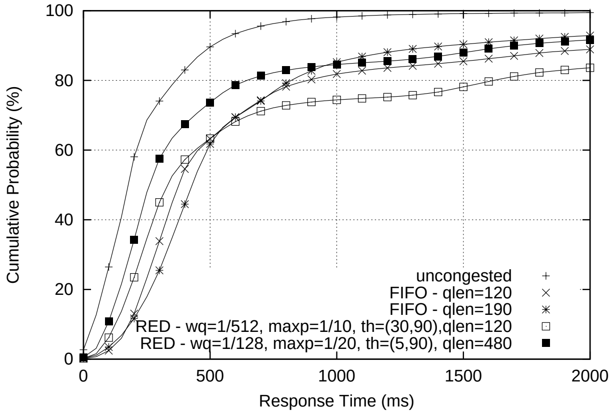

Figure 22b: FIFO and RED at 98% load.

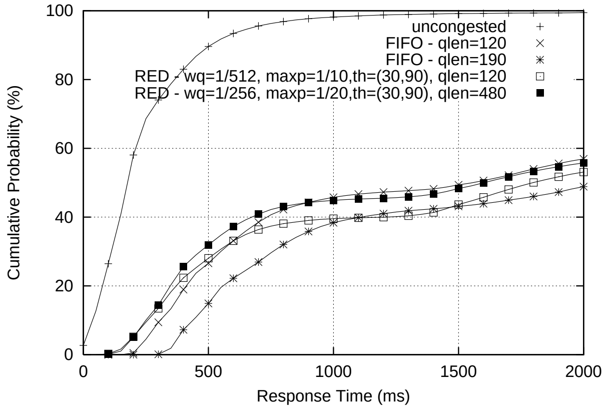

Figure 22c: FIFO and RED at 110% load.

8. Conclusions and Future Directions

Based on our experiments we summarize our conclusions as follows. Contrary to expectations, for offered loads near or below the levels of link saturation (90% or less), there is little difference in end-to-end response times between the best-tuned RED and tail-drop FIFO configured with 1-2 times the bandwidth-delay product in buffer space. Tuning of the RED parameters generally produces little gain (or loss) in response time performance, however, as illustrated in Figure 16a, one can use plausible values for certain RED parameters and produce poorer performance.

At offered loads that approach link saturation (above 90%), RED can be carefully tuned to yield performance somewhat superior to properly configured tail-drop FIFO. The difference is probably significant only between 90% and 100% loading as response times degrade so rapidly above this level that any "improvement" from tuning RED (or FIFO) is, at best, a second-order effect. Moreover, at loads above 90%, response times are more sensitive to the actual values of RED parameters. In particular, there is greater down-side potential from choosing "bad" parameter values as illustrated in Figure 16b. This is significant because parameter settings that outperformed FIFO were arrived at only through extensive trial-and-error experimentation. It was also the case that the RED parameters that provide the best link utilization at this load produce poorer response times.

In general we observed a complex trade-off between choosing parameters that improve response time for short responses (those consisting of only a few TCP segments) and those that improve response times for longer responses. We have chosen to favor those parameter settings that improve performance for the largest fraction of responses, and hence have focused on improving response times for the shorter responses.

Qualitatively these conclusions imply that providing adequate link capacity (utilization less than 90%) is far more important for Web response times than tuning queue management parameters. If one decides to deploy RED for any reason, response times for Web-dominated traffic are not likely to be impacted positively and, unless careful experimentation is performed, response times can suffer. Given the current lack of a widely-accepted analytic model for RED performance or field-tested engineering guidelines for RED deployment and the complexity of setting RED parameters, there seems to be no advantage to RED deployment on links carrying only Web traffic.

In applying these conclusions, there are some limitations of this study that should be considered.

- We used packet-drops as the only "marking" behavior of RED. Explicit marking by RED for ECN-capable TCP implementations is likely to produce better results.
- We examined only HTTP 1.0 protocols. The interaction of RED with a mix of HTTP 1.0 and HTTP 1.1 traffic should also be analyzed.
- 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. Congestion on both paths on a full-duplex link and over multiple router hops, should also be considered.

Removing these limitations to produce a broader perspective on RED behavior is the central theme of our ongoing networking experiments.

9. References

[1] M. Allman and V. Paxson, *On Estimating End-to-End Network Path Properties*, Proceedings of SIGCOMM '99, September 1999, pp. 263-274.

[2] F. Anjum and L. Tassiulas, *Balanced-RED: An Algorithm to Achieve Fairness in the Internet*, *http://www.isr.umd.-edu/TechReports/ISR/1999/TR_99-17/TR_99-17.phtml*.

[3] G. Banga and P. Druschel, *Measuring the Capacity of a Web Server*, Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS), December 1997, pp. 61-71.

[4] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, & L. Zhang, *Recomedations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[5] P. Barford and M. E. Crovella, *Generating Representative Web Workloads for Network and Server Performance Evaluation*, in Proceedings of Performance '98/ACM SIGMETRICS '98, 1998, pp. 151-160.

[6] M. Crovella and A. Bestavros, *Explaining World Wide Web Traffic Self-Similarity*, TR-95-015, Boston University Computer Science Department, Revised, October 12, 1995.

[7] *http://adm.ebone.net/~smd/red-1.html*

[8] *http://www.iet.unipi.it/~luigi/ip_dummynet/*

[9] *ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail*

[10] *http://www.research.att.com/~anja/w3c_webchar/*

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

[12] W. Feng, D. Kandlur, D. Saha, K. Shin, *Blue: A New Class of Active Queue Management Algorithms*, University of Michigan Technical Report CSE-TR-387-99, April 1999.

[13] 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.

[14] S. Floyd, *TCP and Explicit Congestion Notification*, ACM Computer Communication Review, vol. 24 no. 5, October 1994, pp. 10-23.

[15] *http://www.aciri.org/floyd/REDparameters.txt*

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

[17] D. Lin and R. Morris, *Dynamics of Random Early Detection*. Proc. SIGCOMM '97, September 1997, pp. 127-138.

[18] B. Mah. *An Empirical Model of HTTP Network Traffic*, Proceedings of INFOCOM ‘97, April 1997, pp. 592-600.

[19] M. May, J. Bolot, C. Diot, and B. Lyles, *Reasons not to deploy RED*, Proc. IWQoS’99, London, March 199, pp. 260-262.

[20] M. May, T. Bonald, and J. Bolot, *Analytic Evaluation of RED Performance*, Proc. INFOCOM 2000, March 2000, pp. 1415-1424.

[21] *http://www.netstat.net/*

[22] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud’hommeaux, H. Lie, C. Lilley, *Network Performance Effects of HTTP/1.1, CSS1, & PNG*, Proc. SIGCOMM '97, September 1997, pp. 155-166.

[23] T. Ott, T. Lakshman, and L. Wong, *SRED: Stabilized RED*, Proceedings IEEE INFOCOM '99, March 1999, pp. 1346-1355.

[24] *http://null0.qual.net/brad/papers/reddraft.hm*, 1998 (link now broken).

[25] K. Thompson, G. Miller, and R. Wilder, *Wide-Area Internet Traffic Patterns and Characteristics*, IEEE Network, vol. 11 no. 6, November/December 1997, pp. 10-23.

[26] C. Villamizar and C. Song, *High Performance TCP in ANSNET*, ACM Computer Communications Review, vol. 24 no. 5, October 1994, pp. 45-60.

Appendix A
Details of the Experimental Configuration

TCP implementation characteristics in FreeBSD 2.2.8

The laboratory network used to emulate this configuration is a collection of Intel architecture machines running FreeBSD 2.2.8. All machines were configured identically. As configured on our experimental systems, TCP has a default window size in of 16K bytes. The TCP implementation supports Reno congestion control and does not include SACK or new-Reno functions. It supports RFC 1323 (high performance — not enabled in our experiments), RFC 1122 (delayed ACKs), and the Nagle algorithm. It does not support RFC 2414 (larger initial window) or RFC 2481 (ECN). The bugs from RFC 2525 (4.2 BSD) are fixed. The default initial *ssthresh* is 1MB and the behavior at *cwnd* equal to *ssthresh* is slow-start.

Round trip times

We use the *dummynet* [7] component of FreeBSD to configure in-bound packet delays on the end systems to emulate different round-trip times between each paring of a browser machine and a server machine. Table 1 shows the different round-trip times between each paring of browser machine and server machine. The values in this table were taken from measurement data obtained at the *NetStat.net* web site (*http://www.netstat.net*). Each value is the mean of 10 samples taken on different days for the reported round trip times between one of the probe-destination pairs in the NetStat data in July of 1999. The 49 probe-destination pairs were chosen to represent a mix of Internet "distances" in the continental U.S.

A given entry in Table 1 represents the minimum round-trip time experienced by an arbitrary TCP connection between a specific pair of client and server machines in our experiments (assuming no delays in the two routers). Given that the distribution of TCP connections over pairs of machines is designed to be approximately uniform in our experiments, the mean minimal round-trip time for all TCP connections sharing the network is the average of all entries in Table 1. This value is approximately 79 milliseconds.

Web-like traffic generation

The traffic that drives the experiments described here is based on a model of web browsing developed by Mah [18]. The elements that have the most pronounced effects on generated traffic are the size (bytes) of server responses, the number of requests (files) necessary to download a page (including all embedded references), and the user "think" time between successive page requests. Mah found that the median response sizes in four sub-samples of his trace data ranged between 1.5 KB and 2.2 KB while the means ranged from 8.3 KB to 10.6 KB, characteristics that are consistent with heavy-tail distributions. He also found that the distributions of response sizes above 1 KB could be modeled as Pareto distributions with parameter $\alpha = 1.04$ to 1.14. Response sizes are actually further characterized depending on whether the response is to a request for the primary page or an embedded reference. Mah found that the distributions of primary page sizes have a different Pareto model ($\alpha = 0.85$ to 0.97) than sizes of embedded references model ($\alpha = 1.12$ to 1.39). User think times between page requests were found to have medians that ranged between 14 and 16 seconds with means ranging from 837 to 1,916 seconds, again characteristics consistent with heavy-tail distributions (although Mah did not report any attempt to fit the empirical data to a model such as Pareto). The mean number of requests (files) per page ranged between 2.8 and 3.2 with a median of 1. More recent empirical data and models reported by Barford, *et al.*, [4] confirms that these characteristics of Web traffic are well-modeled by heavy-tailed distributions. The sizes of client request messages are relatively small (medians of 231-244 bytes and means of 301-356 bytes) and do not contribute greatly to the overall network loads.

Table 1: Round trip times in milliseconds between pairs of server machines (rows) and client machines (columns).

	brain	**taz**	**tweetie**	**howard**	**lovey**	**speedy**	**petunia**
goddard	81	105	64	64	67	147	114
wako	126	137	47	53	41	86	114
floyd	33	42	40	114	112	117	108
goober	35	45	95	100	31	100	116
thelmalou	105	92	78	41	53	109	66
roadrunner	85	112	38	83	55	8	41
yako	124	87	101	87	95	7	61

Appendix B

Tables of Result Data

Additional FIFO Results — Choice of queue length

Table 1 provides additional data on the performance of tail-drop FIFO queuing for 5 choices of queue length and 4 load levels. These results confirm that our selection of queue sizes of 120-190 represent reasonable tradeoffs for response times without significant loss of link utilization or high drop rates.

Table 1: FIFO results.

Load %	Queue Length	KB per sec.	% drops	Mean queue	Median resp. (ms)	% ≤ 1 sec.	1<%≤2 sec.	2<%≤3 sec.	% > 3 sec.
80	30	992	1.1	6.5	246	92.2	4.5	2.1	1.3
80	60	980	0.3	11.7	248	95.5	2.9	0.9	0.7
80	120	990	0.1	22.7	264	95.8	3.0	0.6	0.6
80	190	992	0.0	27.7	273	95.5	3.3	0.6	0.6
80	240	981	0.0	25.8	265	95.9	3.1	0.5	0.5
90	30	1107	2.2	9.9	258	87.7	6.2	3.7	2.5
90	60	1130	0.9	20.0	266	92.4	4.2	2.0	1.3
90	120	1164	0.3	40.3	298	93.7	4.0	1.3	1.0
90	190	1106	0.2	66.6	361	92.1	5.4	1.3	1.2
90	240	1179	0.3	85.7	397	89.9	6.8	1.6	1.7
98	30	1163	6.7	16.6	329	71.8	11.1	8.5	8.6
98	60	1177	6.2	41.6	375	73.5	9.3	7.8	9.4
98	120	1169	3.1	84.8	421	81.2	7.5	5.2	6.1
98	190	1166	1.3	119.2	478	84.2	8.5	3.3	4.0
98	240	1167	1.4	154.3	555	80.0	11.3	3.7	5.0
110	30	1183	18.9	22.6	1538	44.6	14.5	11.1	29.8
110	60	1189	16.4	52.4	1440	47.7	13.3	11.7	27.4
110	120	1188	17.0	112.3	1600	45.1	11.3	12.9	30.6
110	190	1188	19.3	183.0	2156	37.3	10.8	14.7	37.0
110	240	1188	16.5	231.7	1917	38.7	12.3	13.9	35.0

Additional RED Results — Choice of min_{th} and max_{th}

Here we present additional data to support our choice of values of min_{th} and max_{th} that provide the best performance for our HTTP workloads. In all the experiments reported here we fixed max_p at 0.10, w_q at 0.002 (actually 1/512), and *qlen* at 480.

Performance with max_{th} = 3 x min_{th}

Table 2 presents the performance of RED for 5 pairs of (min_{th}, max_{th}) and 3 load levels. In these experiments the ratio between min_{th} and max_{th} is fixed at 3 as per the RED guidelines [14]. These results (combined with those in the following section) confirm that our selection of queue sizes of 120-190 represent reasonable tradeoffs for response times without significant loss of link utilization or high drop rates. For our subjective measures of performance presented in Section 5, the best overall response time performance is obtained with values for (min_{th}, max_{th}) of (30, 90) or (60, 180).

Table 2: RED results with $max_{th} = 3 \times min_{th}$.

Load %	(min_{th}, max_{th})	KB per sec	% drop	Mean queue	Median resp. (ms)	% ≤ 1 sec.	1<%≤2 sec.	2<%≤3 sec.	% > 3 sec.
90	5,15	1068	3.2	7.1	257	83.6	8.0	4.8	3.6
90	15,45	1088	2.0	14.7	264	88.1	5.9	3.4	2.6
90	30,90	1079	0.8	20.2	266	92.5	4.3	2.0	1.3
90	60,180	1095	0.5	35.5	290	93.1	4.1	1.6	1.2
90	120,360	1094	0.1	53.8	325	93.7	4.4	1.0	0.9
98	5,15	1135	15.2	11.6	680	51.6	14.9	10.2	23.3
98	15,45	1158	5.8	24.0	338	73.3	10.5	7.9	8.3
98	30,90	1164	4.1	39.4	345	79.2	8.2	6.3	6.3
98	60,180	1178	2.4	69.1	384	83.2	7.1	5.0	4.7
98	120,360	1182	3.1	147.4	554	75.6	10.1	6.0	8.3
110	5,15	1147	24.0	12.6	1936	36.9	14.0	9.8	39.3
110	15,45	1175	23.4	36.1	1957	37.1	13.5	10.3	39.1
110	30,90	1187	19.7	76.0	1850	39.4	12.9	12.1	35.5
110	60,180	1187	17.9	157.6	2119	37.9	10.5	15.5	36.1
110	120,360	1188	15.5	303.4	2470	31.6	14.0	14.6	39.7

Varying the ratio of min_{th} to max_{th}

Tables 3 and 4 provide data from the experiments that vary the ratio between min_{th} and max_{th}. In Table 3 max_{th} is fixed at 90 and four values of min_{th} are considered. In Table 4 min_{th} is fixed at 30 and four values of max_{th} are considered.

Table 3: RED results for different values of min_{th} with max_{th} fixed at 90.

Load %	(min_{th}, max_{th})	KB per sec	% drop	Mean queue	Median resp.(ms	% ≤ 1 sec.	1<%≤2 sec.	2<%≤3 sec.	% > 3 sec.
90	5, 90	1088	1.6	12.8	264	89.2	5.7	3.1	2.0
90	15, 90	1104	1.4	17.7	272	89.9	5.3	2.9	1.9
90	30, 90	1079	0.8	20.2	266	92.5	4.3	2.0	1.3
90	60, 90	1079	0.6	27.6	282	93.2	4.0	1.6	1.2
98	5, 90	1149	4.2	24.3	315	78.8	9.0	6.4	5.9
98	15, 90	1161	4.2	30.7	329	78.8	8.6	6.5	6.1
98	30, 90	1164	4.1	39.4	345	79.2	8.2	6.3	6.3
98	60, 90	1171	4.5	54.8	379	78.4	7.5	6.7	7.4
110	5, 90	1188	19.7	75.0	1912	38.0	13.3	12.5	36.2
110	15, 90	1188	19.9	75.2	1902	38.3	13.2	12.4	36.1
110	30, 90	1187	19.7	76.0	1846	39.4	12.9	12.1	35.5
110	60, 90	1187	21.7	80.4	1992	38.0	12.1	11.8	38.1

Table 4: RED results for different values of max_{th} with min_{th} fixed at 30.

Load %	**(min_{th}, max_{th})**	KB per sec	% drop	Mean queue	Median resp.(ms	% ≤ 1 sec.	1<%≤2 sec.	2<%≤3 sec.	% > 3 sec.
90	30, 45	1088	2.1	20.0	279	87.9	5.7	3.7	2.8
90	30, 60	1106	1.6	22.0	280	89.3	5.3	3.2	2.2
90	30, 90	1079	0.8	20.2	266	92.5	4.3	2.0	1.3
90	30, 180	1092	0.9	25.7	281	92.1	4.3	2.0	1.5
98	30, 45	1144	9.4	30.0	377	69.2	10.0	8.4	12.3
98	30, 60	1162	6.7	36.2	372	72.1	9.7	8.1	10.1
98	30, 90	1164	4.1	39.4	345	79.2	8.2	6.3	6.3
98	30, 180	1187	5.6	72.0	446	71.8	9.5	8.4	10.3
110	30, 45	1179	22.5	37.9	1762	41.5	11.9	9.8	36.8
110	30, 60	1182	22.7	49.9	1908	38.0	13.2	10.7	38.1
110	30, 90	1187	19.7	76.0	1846	39.4	12.9	12.1	35.5
110	30, 180	1187	19.4	158.2	2354	35.3	10.4	15.5	38.7