The Performance of HTTP Traffic Under Random Early Detection Queue Management

Mikkel Christiansen <mixxel@cs.auc.dk>
Department of Computer Science, Aalborg University, Denmark

October 11, 2001

Abstract

In RFC (Request For Comments) 2309 [9] the active queuing mechanism RED has been purposed for widespread deployment on Internet routers. This thesis presents an empirical study of the active queuing mechanism RED where we focus on the question: *How will RED effect HTTP response times and can RED be tuned to optimize these?*

The empirical study is conducted on a laboratory network in which we model a typical scenario in which a router operates and becomes a bottleneck. A realistic HTTP traffic load is provided by a set traffic generator programs. These simulate the behavior of browsing users using an empirical model of HTTP traffic that is both well-founded as well as being widely accepted and used. Response time performance is measured for each request made in the simulation of browsing users, thus providing a detailed insight on the performance experienced by the end-user.

The study consists of finding the optimal RED parameters under different offered loads. Where the offered load describes the average bandwidth utilization produced by the traffic generators on a network with no bandwidth constraints. To determine the impact of using RED we compare the response time performance with our choice of optimal RED parameters with the optimal performance of tail-drop queuing and the performance on the laboratory network without bandwidth constraints.

The results of the study can be summarized as follows:

- Contrary to expectations, compared to tail-drop queuing, RED has minimal effect on HTTP response times for offered loads up to 90% of the link capacity. Response times at loads in this range are not substantially effected by RED parameters.
- Between 90% and 100% load, RED can be carefully tuned to yield performance somewhat superior to tail-drop queuing, however, response times are quite sensitive to the actual RED parameter values selected.
- In heavily congested networks, RED parameters that provide the best link utilization produce poorer response times.

In total, we conclude that for links carrying Web traffic, RED queue management appears to provide no clear advantage over tail-drop for end-to-end response times.

Acknowledgements

Foremost I would like to thank the DIRT research group at University of North Carolina at Chapel Hill, in particular:

- Kevin Jeffay and Don Smith for excellent supervision,
- Mark Parris for lots of useful discussions and lab support,
- Jan Martin Borgerson and Don Smith for the original version of the HTTP traffic generator, and
- David Ott for delay measurements, the router-monitor tool, and lab support.

During the Summer of 1998 I arrived at Chapel Hill for a four month visit at the Distributed Real-Time Systems group at UNC. I was received with much openness and within these four months I got deeply involved in the ongoing research activities, and especially the work on studying RED queue management. As a result I continued this work after going back to Denmark, and only though numerous telephone conferences, E-mail exchanges, and several short term visits has is been possible to continue the work on the study.

I would also like to thank my advisor Arne Skou for enthusiastic support, Eric Jul for establishing the contact to UNC, and Anders P. Ravn for commenting on my drafts. My thanks also goes to Emmanuel Fleury commenting on my drafts on Chapter 3, and also thanks to Brian Nielsen for commenting on the final draft of this thesis.

Last but not least i would like to thank my family: Alfred, Ida, and Janne for their unconditional support.

Financial support has been provided through a scholarship from Aalborg University.

Dansk Sammenfatning[1]

Denne afhandling dokumenterer en undersøgelse af den aktive kø-mekanisme Random Early Detection (RED). Vi fokuserer specifikt på at vurdere, om denne kan forbedre de svartider, en bruger oplever, når web sider hentes via internettet. Herunder er vi specielt interesserede i, hvorledes parametrene til RED skal justeres for at opnå optimale svartider.

RED er en aktiv kø-mekanisme, der kan anvendes på internet-routere. En internet-router er en såkaldt *store-and-forward* router, idet at pakker kan opbevares i en buffer, indtil der er båndbredde til rådighed på destinationsnetværket. Traditionelt bliver denne buffer administreret efter *first come first serve* princippet og i tilfælde af, at bufferen bliver mættet, vil nyankomne pakker blive tabt. Denne form for administration af bufferen kaldes *tail-drop*.

RED er en algoritme udviklet som et alternativ til tail-drop med det formål at øge den overordnede netværksydelse i tilfælde af mætning. Intuitivt er ideen, at RED algoritmen skal detektere en mætning af netværket allerede i opløbet, for derved tidligt at kunne sende et mætningssignal til senderne. Senderne reagerer på et mætningssignal ved at reducere transmissionshastigheden, og da dette kommer tidligt, vil en mætning af bufferen undgås eller mætningsperioden vil blive reduceret.

Helt konkret er RED en algoritme, der udvælger hvilke pakker der skal indsættes i bufferen, og hvilke pakker, der skal tabes. Tabte pakker opfattes som et mætningssignal af sendere, og herved kan pakketab reducere belastningen af netværket. Selve udvælgelsen er baseret på en vægtet gennemsnitlig kølængde, som afspejler belastningen af routeren over en længere periode.

I RFC 2309 [9] tilråder forfatterne, at RED generelt anvendes på internet-routere. Rådet er baseret på de hidtidige empiriske undersøgelser, som på det tidpunkt var til rådighed. Disse undersøgelser viser, at anvendelsen af RED på internet-routere vil forbedre ydelsen på netværket eller i det mindste ikke forringe den.

Det er vores opfattelse, at det empiriske materiale, der er grundlaget for dette råd, ikke er fyldestgørende. For eksempel er det et problem, at RED algoritmen tilføjer kompleksitet til netværket. Dette sker, idet algoritmen har flere parametre i forhold til tail-drop, og disse skal vælges, inden algoritmen kan tages i brug. Grundet den meget dynamiske adfærd af datatransmissioner på internettet, er det en ikke-triviel opgave at afgøre om et givet sæt parametre er det optimale valg. Endvidere kan de optimale parametre være afhængige af den situationen, hvorunder algoritmen opererer.

I denne afhandling fokuserer vi netop på det problem, at afgøre hvilken indflydelse det har at anvende RED på internet-routere. Vi gør dette ved at stille spørgsmålet: *Hvordan vil RED påvirke HTTP svartider, og kan RED blive justeret til at forbedre disse?*

HTTP er den protokol, som anvendes til af udveksle objekter imellem en Web browser og en web server. Vi har valgt at beskæftige os med HTTP-svartider, da

[1] in danish

Web-applikationen er den oftest anvendte på internettet. HTTP-trafik udgør 60-80% af båndbredde forbruget på internettet i dag [65]. Det at browse på internettet er præget af en høj grad af interaktion imellem browser og server. Derfor er HTTP-svartider et centralt begreb, når vi diskuterer netværksydelse.

Vi søger at besvare det stilte spørgsmål igennem en empirisk undersøgelse. Dette gøres ved en række forsøg på et laboratorie-netværk, hvor vi modellerer en typisk situation, hvor en router anvendes og kan være en flaskehals. Trafikken på dette netværk genereres ud fra en empirisk model af Web trafik (HTTP 1.0), som er bredt accepteret og tidligere blevet anvendt i eksperimentelle sammenhænge. Baseret på denne opstilling har vi mulighed for at modellere og reproducere den dynamiske adfærd af internetprotokoller i et kontrolleret miljø, således at vi har mulighed for at studere RED under forskellige forhold, konfigureret med forskellige parametre.

Ved brug af vores laboratorieopstilling har det været muligt af foretage en detaljeret undersøgelse af, hvorledes RED påvirker HTTP svartider under en række forskellige omstændigheder. Specielt har vi undersøgt hvorledes RED påvirker svartider under forskellige belastningsniveauer, baseret på båndbreddekapaciteten i flaskehalsen. Disse undersøgelser er blevet sammenlignet, dels med svartider målt ved brug af tail-drop algoritmen, og dels med svartider målt på et netværk, som ikke er begrænset af båndbredde.

Resultaterne af vores undersøgelse viser, at RED har forskellig indflydelse på svartiderne afhængigt af dens belastningsniveau. For trafikbelastninger under 90% af båndbredde-kapaciteten er svartiderne forholdvist tæt på, hvad vi oplever uden begrænsninger i båndbredde-kapacitet. For trafikbelastninger, som nærmer sig båndbredde-kapaciteten, dvs. over 90% men under 100% belastning, kan vi justere RED til at give svartider, der er bedre end det, vi har fundet med tail-drop. Denne observation er dog kun aktuel, når belastningen er imellem 90% og 100%. Når belastingen overstiger 100% af kapaciteten, forringes svartiderne hurtigt, og vi observerer ingen forskel imellem brugen af RED og tail-drop.

En generel observation for belastninger under 100% af båndbredde kapaciteten er, at RED parametre, der generelt virker fornuftige, kan resultere i en forværring af svartider. Dette er specielt problematisk, da vi jo netop kun fandt de optimale parametre igennem en systematisk afprøvning af forskellige parameter værdier, hvilket normalt ikke er muligt. En anden vigtig observation er, at de RED parametre, som giver den højeste udnyttelse af båndbredde, og det procentvist laveste tab af pakker, resulterer i en forværring af svartider. Derfor er disse, ellers hyppigt anvendte ydelsesparametre, ikke direkte anvendelige til at vurdere ydelsen af svartider. Til sidst kan vi nævne, at vi har observeret, at det at vælge optimale parametre er en kompleks vægtning mellem at optimere for de kortest mulige svartider, og det at få færre svar med lange svartider.

Helt overordnet viser vores undersøgelse, at det er langt vigtigere for web-svartider at tilstrækkelig båndbredde er til rådighed, frem for at have de optimale RED parametre. I det tilfælde, at RED anvendes i et netværk domineret af web-trafik, skal dette omhyggeligt forberedes igennem en lang række forsøg hvor forskellige RED parametre afprøves, for derved at forebygge en eventuel forværring af svartider.

Contents

Abstract iii

Acknowledgements v

Dansk Sammenfatning (in danish) vii

1 Introduction 1
- 1.1 Networks . . . 1
- 1.2 Routers . . . 3
- 1.3 Congestion Avoidance . . . 4
- 1.4 Random Early Detection . . . 4
- 1.5 Thesis . . . 6
- 1.6 Overview . . . 6

2 Background 7
- 2.1 Random early detection . . . 7
  - 2.1.1 Motivation . . . 7
  - 2.1.2 The Algorithm . . . 9
  - 2.1.3 Parameters . . . 12
  - 2.1.4 Aggressive flows . . . 13
  - 2.1.5 Summary . . . 14
- 2.2 Related Work . . . 15
  - 2.2.1 The Original Evaluation . . . 15
  - 2.2.2 An Empirical Evaluation . . . 18
  - 2.2.3 Alternative Queuing Mechanisms . . . 19
  - 2.2.4 Analytical Evaluation . . . 21
  - 2.2.5 Feedback Control . . . 22
  - 2.2.6 Pilot Tests . . . 22
  - 2.2.7 Summary . . . 23
- 2.3 Summary . . . 23

3 Traffic Generation 25
- 3.1 Related Work . . . 25
- 3.2 Web Traffic . . . 26
  - 3.2.1 Self-similarity . . . 27
  - 3.2.2 Long-range Dependence and Heavy-tailed Distributions . . . 29
  - 3.2.3 ON/OFF model . . . 29
  - 3.2.4 Summary . . . 30
- 3.3 The Mah Traffic Model . . . 31
  - 3.3.1 Behavior . . . 31
  - 3.3.2 Distributions . . . 32
- 3.4 Implementation . . . 38

3.4.1 Performance Measurements . . . 38
3.4.2 Protocol . . . 39
3.4.3 Multiplexed I/O and TCP Configuration . . . 40
3.4.4 Modifications and Bugs . . . 40
3.5 Test of Traffic Generator . . . 40
3.5.1 Network Setup . . . 41
3.5.2 Bursty Traffic . . . 41
3.5.3 Bottleneck and Independence Tests . . . 43
3.5.4 Calibration . . . 45
3.5.5 Summary . . . 46
3.6 Conclusion . . . 47

4 Network Design and Experimental Procedures 49
4.1 Network Design . . . 49
4.1.1 Network Model . . . 49
4.1.2 Modeling Latency . . . 50
4.1.3 Monitoring . . . 51
4.1.4 Operating System and Protocol Configuration . . . 52
4.2 Experimental Procedures . . . 52
4.3 Performance Evaluation . . . 53
4.3.1 Experiment: unconstrained (100Mbps) network . . . 53
4.3.2 Experiment: constrained network with RED . . . 54
4.3.3 Comparing Response Time Performance . . . 56
4.4 Summary . . . 57

5 Tuning RED 59
5.1 Tail-Drop Results . . . 59
5.1.1 Methodology . . . 59
5.1.2 Results . . . 60
5.2 RED Results . . . 62
5.2.1 Methodology . . . 62
5.2.2 Results . . . 63
5.3 Comparing Tail-drop and RED . . . 68
5.4 Summary . . . 69

6 Analysis of RED Response Times 73
6.1 Impact of Packet Drops/Retransmissions . . . 73
6.1.1 Methodology . . . 73
6.1.2 Results . . . 74
6.1.3 Summary . . . 77
6.2 Congestion and Web-like Traffic . . . 77
6.2.1 Methodology . . . 78
6.2.2 Results . . . 79
6.2.3 Summary . . . 84
6.3 Summary . . . 85

7 Conclusion and Further Work 87
7.1 Results . . . 87
7.2 Further Work . . . 88
7.3 Summary . . . 89

A Laboratory Network 91
A.1 Network Diagram . . . 91

B Experiments with Tail-drop and RED **93**
B.1 Experiments . . . 93
B.1.1 Tail-drop Experiments . . . 93
B.1.2 RED Experiments . . . 93
B.2 Tail-drop Summary . . . 94
B.3 RED summary . . . 100
B.3.1 Experiments with min_{th} and max_{th} . . . 100
B.3.2 Experiments with max_{th} . . . 104
B.3.3 Experiments with min_{th} . . . 107
B.3.4 Experiments with w_q and max_p . . . 109
B.3.5 Experiments with *qlen* . . . 111
B.3.6 "Good" and "Bad" Parameter Settings . . . 113
B.4 Comparing RED and Tail-drop . . . 116

C Alternate Queueing **119**

Bibliography **121**

Chapter 1

Introduction

Random Early Detection (RED) is an active queueing mechanism designed for congestion avoidance in packet switched networks. The queueing mechanism detects incipient congestion by computing the average queue size of a router queue. When the average queue size exceeds a threshold, the router signals congestion with a probability, that is a function of the average queue size. RED was first described in [29] and has later been recommended for deployment on Internet routers in the RED manifesto [9].

In this dissertation we present an empirical study that evaluates the effect of using RED on Internet routers carrying Web traffic.

In the following section we give a brief introduction to packet switched networking routers and congestion avoidance, in particular. This is followed by an introduction to the RED queuing mechanism. Based on these concepts we are able to state the thesis for this dissertation.

1.1 Networks

The most popular application on the Internet is currently the World Wide Web (Web). Recent studies shows that Web traffic accounts for 60-80% of the total traffic carried on backbone links [65].

Web servers and browsers communicate using the Hyper Text Transfer Protocol (HTTP) [8, 22]. HTTP is a generic, stateless, object oriented protocol, that, among other things, allows clients to request a specific file from a Web server. On the Internet, HTTP communication generally takes place using the TCP/IP protocol suite. Figure 1.1 illustrates how Web servers and browsers are connected to the Internet. A client requests a Web page from a server by sending a HTTP request to the server. The request is transferred to the server through network links that are shared equally among all hosts on the network.

The Internet protocol (IP) [58, 63] provides a connectionless delivery service for transmitting blocks of data as packets from sources to destinations on a unreliable network. This means that IP provides the limited but important functionality of allowing segments of data to be sent from one host to another independent of the properties of the physical transmission links.

The Transmission Control Protocol (TCP) [34, 63, 2] builds on IPs delivery service by allowing applications to establish a reliable virtual-circuit between two hosts on the Internet, also referred to as a TCP connection. Once a TCP connection is established, the data sent between the hosts is guaranteed delivery, meaning that ordering of packet and retransmission of lost packets is transparent both to the application sending the data, but also to the one receiving. This service is exactly

what the majority of applications using the Internet requires, and as a result TCP is by far the most dominating transport protocol used on the Internet today.

The functionality of TCP is much more far reaching than establishing a reliable connection between two hosts on the Internet. As described, links on the Internet are shared among all hosts on the network. Consequently the cumulative traffic load generated by the hosts can exceed the capacity of a link in the network which is referred to as the link becoming congested. TCP implements algorithms for congestion avoidance that dynamically control the sending rate such that it matches the bandwidths available on the path from the sender to the receiver. Congestion avoidance is extremely important in packet switched networks, because it provides the mechanism for preserving the bandwidth and essentially avoiding congestion meltdown [34] during high loads. We return to the TCP congestion avoidance mechanism in Section 1.3.

When a file is transferred between a Web server and a browser using HTTP over TCP/IP, then first of all a virtual connection is established between the two hosts. The file is then broken into smaller segments of data that can be transferred in a separate packets. Each packet consists of a TCP/IP protocol header and a payload with a segment of the file being transferred. The TCP header contains information such as which virtual connection the packet belongs to and how the payload is positioned relative to the payloads of other packets, such that the file later can be reassembled. The IP header contains information such as the source and destination address of a packet.

The stream of packets produced by a TCP connection is referred to as a flow. TCP flows are all treated equally in the network and expected to be well behaved, that is, conform to the TCP protocol specifications [2].

The general behavior of traffic on Internet links has been subject to much study, see among others [35, 10, 38, 13, 40, 19]. The interest lies in providing models of traffic that can be used for designing experiments in which new protocols or algorithms can be tested. A central result from these studies of traffic behavior is that the traffic generated by the Web shows evidence of probabilistic fractal like behavior [12]. Probabilistic fractal like behavior describes a packet arrival process that is highly fluctuating and unpredictive independent of the timescale at which it is viewed. Such an arrival process can have a negative impact on the network performance, because bursts of traffic can cause periods of congestion even though the general load on the network is low.

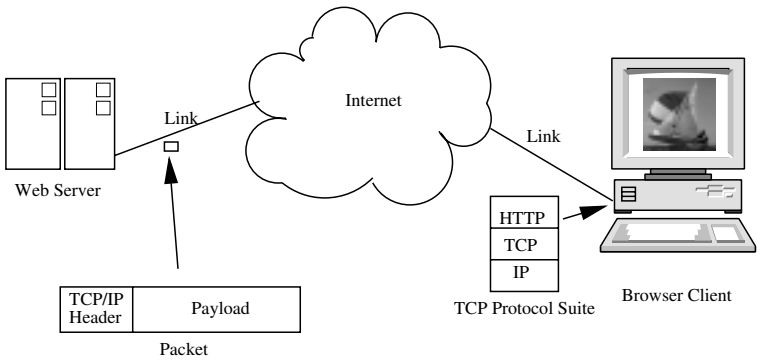


Figure 1.1: *Internet Connection: A Web server and a browsing client are connected to the Internet through physical links. Each host has a protocol suite for communicating and data is transmitted as packets.*

1.2 Routers

Routers are computers that connect to more than one network, and that provides a gateway through which packets can be routed between networks. Internet routers are *store and forward routers* where packets are stored in a buffer for the designated outgoing link until bandwidth becomes available on the link and the packet can be forwarded. Using this technique, routers have the ability to move packets from one network to another independent of transmission properties of the networks such as different delay and bandwidth properties. For instance, routers can move packets between a network with a 100Mbps Ethernet link and a network with a 56Kbps modem link.

This flexibility of store and forward routers, in combination with TCP congestion avoidance algorithms, provides a very strong technology for building networks consisting of different transmission technologies. However, this also means that the router may become a bottleneck in a transmission between hosts on the network.

A router, see Figure 1.2, consists of a number of network interface cards (NICs), these cards receive and send packets on their respective networks. To each NIC there is an associated buffer for outbound traffic. Traditionally this buffer is managed using First Come First Serve (FCFS) scheduling. If the buffer overflows, then arriving packets will be dropped. This combination of FCFS buffer scheduling and drop selection is generally referred to as tail-drop queueing.

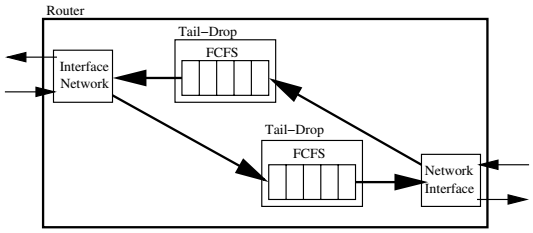


Figure 1.2: *An Internet Router.*

The idea of RED and other active queue management (AQM) mechanisms is to modify the tail-drop mechanism. The modification is done by adding an algorithm for selecting which packets are dropped and which packets are buffered. This is illustrated in Figure 1.3. Compared to tail-drop queuing, adding an active queue management algorithm allows us to improve the overall performance in the network by adding a specific algorithm for selecting which packets should be dropped and when.

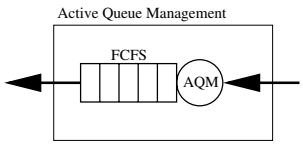


Figure 1.3: *A Buffer with Active Queue Management.*

1.3 Congestion Avoidance

TCP congestion avoidance is a protocol mechanism that preserves bandwidth by continuously adjusting the transmission rate based on feedback from the network. In the following we give an informal introduction to TCP congestion avoidance principles, while we refer to [34, 63] for a detailed descriptions.

A packet sent to a receiver carries a sequence number which uniquely places the packet within a flow. Upon reception of a packet the receiver acknowledges the received packet by sending an acknowledgement back to the sender with the sequence number of the received packet. Therefore, if an acknowledgement is missing, the sender times out, and interprets this as a packet drop. Packet drops are interpreted as an indication of congestion, and TCP reacts to this by reducing the rate at which packets are injected into the network.

The transmission rate of a TCP sender is controlled using the sliding window protocol [63]. The sliding window protocol works by only allowing a sender to have a "window" of data in the network at any time. Then as acknowledgements arrive from the receiver, indicating that the data has been received, the window slides forward and the sender can send more data. By changing the size of the window we can either decrease or increase the sending rate.

TCP congestion avoidance mechanism continuously changes the size of the congestion window of TCP based on the stream of acknowledgements and detections of packet loss. This continuous adjustment of the transmission rate ensures that the sender receives the highest possible amount of bandwidth, while allowing the bandwidth to be shared almost equally between a changing number of flows.

To be more exact, the congestion avoidance mechanism is a mix between three mechanisms: 1) Slow Start, 2) Additive Increase Multiplicative Decrease (AIMD), and 3) retransmission timers.

Slow start is used during the initial phase of a connection for quickly probing the network for the amount of bandwidth available for the transfer. Once the connection passes the initial slow start phase then AIMD is used. Here the sender continues to send at a steady rate additively increasing the size of the transmission window until congestion is detected. When congestion is detected then the transmission window is multiplicatively decreased i.e. cut in half, thus reducing the transmission rate. This reduction in the transmission is referred to as TCP "backing off".

The retransmission timer algorithm describes the timeout period from when packet is sent until the sender interprets a missing acknowledgement as a packet drop. However drop detection based on retransmission timeout is not very efficient due to the necessary conservative estimation of the propagation delay. Therefore, TCP has been extended with a fast retransmit/recovery mechanism [2]: If a receiver receives a packet with an out of order sequence number, then instead of sending a new acknowledgement, the receiver sends a duplicate of the previous acknowledgement. If the sender receives three or more duplicate acknowledgements, then it is interpreted as an indication that the packet following the multiple acknowledged packet has been lost. As in the case of loss detection by timeout, the sender will reduce its transmission rate and retransmit the lost segment.

1.4 Random Early Detection

The principle goal of Random Early Detection (RED) is to act as a congestion avoidance mechanism. In practice the idea is to replace the traditional tail-drop queuing mechanism with a more complex mechanism that supports the TCP congestion avoidance mechanisms, such that periods of congestion on routers can be avoided or the duration reduced.

The operation of RED is illustrated in Figure 1.4. RED uses a weighted average queue length that is computed from samples of the buffers instantaneous queue size (*qlen*) at each packet arrival. Figure 1.4 shows how RED reacts slowly to changes in the buffer utilization by showing both the weighted queue length (*wqlen*) and the instantaneous queue length. Therefore, a burst of packets need to be of a certain duration before the weighted average queue length changes significantly.

Based on the weighted average queue size RED operates in three different modes: If *wqlen* is below the minimum threshold (min_{th}) then no packets are dropped, if above the maximum threshold (max_{th}) the router drops all arriving packets (force-drop mode). When *wqlen* is between the thresholds packets are dropped probabilistically (early-drop mode). In early-drop mode packets are dropped with a probability between 0 and a maximum drop probability. The drop probability is a function of the weighted average queue size and the number of packets that have been forwarded since the last drop, thus ensuring that the packet drops are uniformly distributed.

Returning to Figure 1.4, we see that during the entire period, packet drops are all early drops even though there are periods, where the instantaneous queue length increases above the maximum threshold.

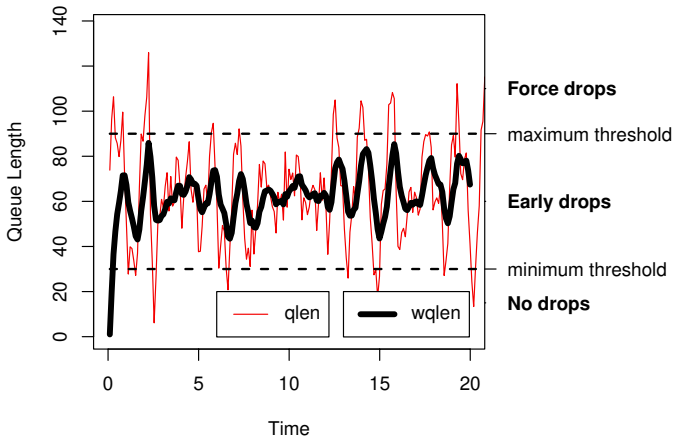


Figure 1.4: *Illustration of RED in operation.*

There are several advantages of using this drop scheme compared to tail-drop queuing. First of all, flows will be affected proportionally to the amount of bandwidth they use, thus high bandwidth flows are more likely to receive congestion notifications than low bandwidth flows. Furthermore, the mechanism maintains a lower average queue size than the traditional tail-drop management, it avoids lock-out where a flow can monopolize the bandwidth, and avoids repeated penalization of the same flow when a burst arrives [29, 9]

RED has been implemented in routers and tested through numerous experiments [9]. Therefore [9] recommends that RED should be widely deployed, based on the argument that all empirical evidence shows that there are seemingly no disadvantages in using RED and numerous advantages. However, RED adds complexity to the issue of fine tuning a network for good performance. Selecting a queue length parameter that ensures optimal performance with tail-drop queuing is already a difficult task. REDs additional four parameters significantly increases the complexity of tuning, even though guidelines exist [25].

1.5 Thesis

In this dissertation we address the problem of determining the impact of using RED on Internet routers. We do this by asking the question: *How will RED effect HTTP response times and can RED be tuned to optimize these?*.

We have selected Web traffic because this is the single most popular class of traffic on the Internet today using 60-80% of the total bandwidth on Internet backbones [65]. Browsing the Web is a highly interactive application where users follow hyperlinks and expect a fairly immediate response. This makes response times central for the users experience, and should therefore be of significant interest to Internet Service Providers that run networks carrying Web traffic.

We have chosen to evaluate the RED mechanism because it was suggested for deployment in the RED manifesto [9], and because it is already supported in much router software.

We approach this question by performing an empirical study. The experiments are done in a network where we are able to model a typical scenario in which a router operates and become a bottleneck. Web traffic is generated using a well understood and accepted empirical model of the traffic generated by users browsing the Web. Based on this we are able reproduce the dynamic behavior of protocols in a closed environment and thus able to study the effect of using RED under different conditions and configured with different parameter settings.

1.6 Overview

In Chapter 2 on page 7 we study background and related work of the topic of this dissertation. We especially study the motivation behind developing RED, and the methods used for evaluating both RED and other active queue management mechanisms. An important aspect in our methodology is traffic generation. We address the issue of generating a realistic traffic load in Chapter 3 on page 25. We argue that realistic traffic generation is important and describe previous work on the topic. Furthermore, we describe the model of Web traffic that we have used along with the implementation of our traffic generator. Finally, we show the results of testing the behavior of the traffic generator. In Chapter 4 on page 49 we go into detail of describing the experimental network used for the evaluations, the procedure for running experiments, and the method we use for comparing experimental results. Chapter 5 on page 59 is the primary result chapter. In this we describe the results of comparing the performance between RED and tail-drop queuing. The overall comparison is followed by a further analysis of RED response times, thus providing an insight on the factors causing the measured changes in performance. In Chapter 7 on page 87 we discuss the consequence of the analysis presented in this thesis and potential further work.

Chapter 2

Background

We start this chapter with a section describing RED including the motivation for developing RED, the algorithm, the parameters, and so forth. This is followed by an introduction of the different approaches used in the literature for evaluating the effects of using active queue management.

We conclude that RED adds additional complexity to tuning Internet routers. Furthermore, there is no empirical evidence displaying the impact that RED has on Web traffic response times, or studies of how to choose parameter settings on routers carrying Web like traffic.

2.1 Random early detection

RED was originally described by Floyd and Jacobson in [28] and was later recommended for deployment on Internet routers in [9], with expectations that it would benefit the overall performance or at least not have any negative effects on performance:

> "All available empirical evidence shows that the deployment of active queue management mechanisms in the Internet would have substantial performance benefits. There are seemingly no disadvantages to using the RED algorithm, and numerous advantages. Consequently, we believe that the RED active queue management algorithm should be widely deployed.
>
> We should note that there are some extreme scenarios for which RED will not be a cure, although it won't hurt and may still help. An example of such a scenario would be a very large number of flows, each so tiny that its fair share would be less than a single packet per round trip time. [9]"

In the following we describe the general motivation and ambition of designing and deploying RED on the Internet, then we describe the algorithm.

2.1.1 Motivation

The primary motivation for developing RED is to provide a router based mechanism for congestion avoidance. Congestion control of Internet traffic is implemented by TCP implementations on the hosts on the edge of the network. However, the hosts have very limited information regarding the general state of the network and therefore has limited information for choosing an appropriate sending rate. The router on the other hand, has very precise knowledge about its state. By developing

active queue mechanisms it may be possible to propagate this knowledge to the hosts by dropping packets early. Essentially this may improve the general ability to choose a suitable sending rate and thus avoid congestion.

The idea was originally proposed by Van Jacobson in 1989 [43], and more recently it has gained momentum resulting in the RED manifesto [9] which recommends widespread deployment of RED on Internet routers. Additional motivation for developing RED was the problems observed with tail-drop queue management, these include:

1. Full Queues,
2. global synchronization, and
3. bias against bursty traffic.

In the following sections we take a closer look at each of the problems observed with using the traditional tail-drop queuing mechanism.

Full Queues

Studies of Internet traffic shows that the packet arrival process is bursty [38, 68, 13]. This means that routers need to be designed for handling bursty traffic by actively avoiding congestion and delays of packets. The bursty traffic causes the *full queues problem* which affects the performance with tail-drop queue management [9]. Tail-drop has no method for limiting the amount of buffer space being used in the queue. Therefore, it is possible for the mechanism to maintain a full (or, almost full) status for long periods of time. If the queue is full or almost full, an arriving burst will cause multiple packets to be dropped.

Multiple packet drops can cause underutilization of the network because multiple drops may cause many hosts to reduce their transmission rates simultaneously. Also, dropping multiple packets may punish bursty flows harder then other flows and thus breaking goal of fairness among flows.

Another effect of the full queues problem is that by allowing the queue to grow large, the average queue size increases, which again causes an additional delay of packets traversing the router [9]. If the queue management mechanism can control the amount of buffer space used, without increasing the total number of packet drops, then the mechanism reduces the overall propagation delay of packets traveling through the router. Thus, the queuing mechanism will provide lower delays which are important for interactive services like the Web.

Global Synchronization

Global synchronization describes the situation where hosts on a network unintentionally become synchronized. The synchronization effect can occur at different levels of network: protocols can become synchronized by reacting to observed behavior simultaneously, and applications can become synchronized.

At the application level, paper [30] studies the effects of periodic routing messages and shows that synchronization effects can be avoided by adding a small delay of random size to the routing timer intervals.

Zhang *et al.* studies and example of synchronization at protocol level is studied in [70]. This case shows that TCP window increase/decrease cycles of separate TCP connections sharing a common bottleneck router using drop tail queue management. The synchronization effect arises when the router is congested for some period and a sequence of packets arriving at the router is dropped. This causes all flows to detect congestion and thus their window increase/decrease cycles can become synchronized. Overall global synchronization cause the traffic in a network to become more bursty, which has a negative impact on the performance in a network.

Bias against bursty traffic

Floyd *et al.* [28] studies the bias against bursty traffic when using tail-drop queuing on routers. The study uses simulations to study traffic phase effects in networks with highly periodic traffic and deterministic routers. The conclusion is that traffic phase effects, combined with the use of tail-drop queuing on routers can result in systematic discrimination against a particular flow. To understand this we need to describe the *traffic phase effect*.

Consider a simple network with a single router through which a sender makes a bulk transfer to a receiver. The packets from the sender are all of equal size and sent at a fixed frequency. The network pipe is completely full, meaning that all buffers on the path from the sender to the receiver are full. Consequently all packets have the same transfer time, thus keeping the system in a stable state. The phase is the time in which there is space available in the buffers of the routers between the departure of on packet and the arrival of the next. Figure 2.1 illustrates the phase by describing the router as two parallel processes *arrival* and *departure* in a message sequence chart. The arrival processes experience a number of arrival events when packets arrive, likewise the departure process have departure events when packets depart from the router buffer. When the buffer is full or can contain only one packet, then send and receive are mutually related. An arrival is always followed by a departure, and a departure is always followed by an arrival, as illustrated by the arrows. The phase is the time during which the router is able to accept an arrive event

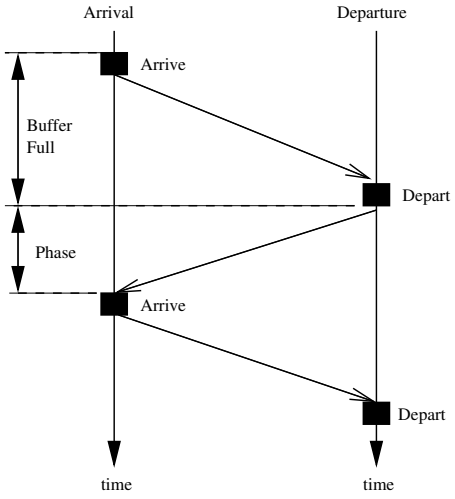


Figure 2.1: *The Traffic Phase Effect.*

Now consider adding a non periodic transfer, such as a telnet connection that can be characterized by a Poisson probability distribution [28]. The packets from the telnet connection will only be allowed in the buffer if they arrive within the phase, otherwise they will be dropped. If the phase is small, the telnet connection may be completely shut out. In contrast, if the phase is large, the telnet connection may always get preference. This effect is referred to as the traffic phase effect.

2.1.2 The Algorithm

The RED algorithm [29] consists of two main parts. The first part estimates the burstiness of the traffic by maintaining a weighted average queue size. The second

part takes the packet drop decision based on the weighted average queue size and the number of packets processed since the previous drop.

Calculation of the weighted average queue size (avg) is done using a low-pass filter which is an exponential moving average (EWMA). This essentially means that avg only changes when changes in the queue size have a certain duration:

$$avg \leftarrow (1 - w_q)avg + w_q \cdot q, \quad \text{where } 0 < w_q \leq 1 \tag{2.1}$$

where q is the instantaneous queue size, and w_q is the weight. If $w_q = 1$, then avg corresponds to the instantaneous queue size.

Periods may occur where the router is idle, i.e. the queue is empty. When a packet arrives to an empty queue, the idle period is taken into account. This is done by estimating the number of packets (m) that *might* have been transmitted since the queue became empty, and using that estimate for estimating the average queue size.

$$m \leftarrow (time - q_{time})/s \tag{2.2}$$

$$avg \leftarrow (1 - w_q)^m \cdot avg \tag{2.3}$$

where $time$ is the current time, and q_{time} is start of the queue idle time. s is a typical transmission time for a small packet.

RED operates in three different packet drop modes depending on the value of the weighted average queue size compared to the threshold values min_{th} and max_{th}. If the weighted average queue size lies below min_{th}, no packets are dropped. If avg lies above max_{th} then all arriving packets are dropped. When avg lies between the two thresholds, RED drops packets probabilistically. The packets dropped while in this mode are referred to as early drops, while packets dropped while $avg > max_{th}$ are referred to as force drops.

The drop probability of packets, while in early drop mode, is described by p_b and p_a. The probability p_b increases linearly from 0 to max_p as a function of the weighted average queue size:

$$p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th}) \tag{2.4}$$

where max_p describes the maximum drop probability and ($min_{th} \leq avg < max_{th}$). The drop probability p_a increases as a function of p_b and the number $count$ of packets that has arrived since the previous drop:

$$p_a \leftarrow p_b/(1 - count \cdot p_b), \tag{2.5}$$

where $count < 1/p_b$. Inclusion of $count$ in the drop probability calculation ensures that packet drops are close to a uniform distribution.

In early drop mode, flows are expected to experience packet drops roughly proportional to the share of the bandwidth being used at the router, simply because sending a higher number of packets through the router increases the chance of experiencing a packet drop due to the uniform distribution of drops. Furthermore, evenly distributing the packet drops may help avoid global synchronization effects where many senders reduce their windows simultaneously, thus increasing the burstiness of the traffic.

It is expected that RED removes the bias against flows that send traffic in bursts. A traditional tail-drop queuing mechanism may overflow when a burst of traffic arrives, resulting in a large percentage of the packets in the burst being dropped. RED, on the other hand, will try to absorb the burst, and avoid dropping a sequence of packets. However, these advances are removed if the mechanism is pushed into force drop mode too often.

To complete the description of the RED algorithm we have included a detailed description of the algorithm in Figure 2.1.2.

Saved Variables:
avg: average queue size
q_{time}: start of the queue idle time
$count$: packets since last dropped packet
Fixed Parameters:
w_q: queue weight
min_{th}: minimum threshold
max_{th}: maximum threshold
max_p: maximum drop probability
s: is a typical transmission time for a small packet
Other:
q: current queue size
$time$: current time

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

Figure 2.2: *Detailed describing the RED algorithm[29].*

2.1.3 Parameters

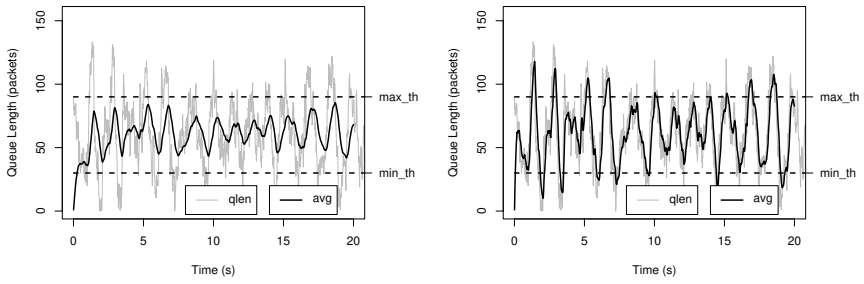
From our description of RED we can see that one has to choose several parameters for a RED router. These are the weight, w_q, the thresholds min_{th} and max_{th}, the maximum drop probability max_p, and additionally the total amount of buffer space allocated for the algorithm, which was not mentioned in the description of RED. Guidelines exists for setting the parameters and can be found in [25].

In the following we discuss the effect of each of these parameters.

The weight (w_q)

The weight is the key parameter for setting the sensitivity or responsiveness of RED to changes in the traffic behavior. If w_q is set too large, then RED may react too fast to transient bursts of traffic, and thus not filter out transient congestion at the router. If w_q is too small, then the weighted average queue length may not reflect the current average queue size, and RED will not be able to detect congestion early. The recommended value of w_q is 1/500.

Figure 2.3 shows the effect of changing w_q from 1/500 to 1/100 on a 10Mbps link. From the plot we can see that using a high w_q value increases the speed at which the algorithm reacts to changes in the instantaneous queue size. However, changing between different values of w_q may not have an impact on the overall number of packets being dropped by the router. The question is rather when to react to congestion and for how long. This is illustrated in the figure where we can see that a small w_q value causes RED to operate between the two thresholds during the entire period, and that a larger value changes the weighted average queue length to values outside the threshold window.



(a) $w_q = 1/500$ (b) $w_q = 1/100$

Figure 2.3: *Two plots showing the effect of w_q on the weighted average queue length.*

Thresholds

The threshold values bound the desired average queue size. When the weighted average queue size reaches the level of min_{th}, RED actively signals congestion thereby trying to limit the queue size. In [25] a general rule is stated that one should choose a low value of min_{th} if possible. However, if the traffic is bursty, one should choose a higher value allowing space for bursts of traffic. max_{th} should be several times larger then min_{th}, so that the window between the two thresholds is large enough for RED to operate. The guidelines for tuning RED says that max_{th} should be two to three times larger then min_{th}.

The optimal setting of the threshold values is mainly a function of link speed and propagation delay. A 10Mbps router is able to forward approximately 1000 packets per second with an average packet size around 1KB. For instance, if the average queue size is 20 packets, the average delay is 20ms per packet. Over short distances 20ms may be a significant contribution to the propagation delay, where as in other situations 20ms may be an almost negligible contribution to the propagation delay. Therefore there is no simple way to choose optimal threshold values.

Maximum drop probability (max_p)

max_p describes how aggressively RED should respond with packet drops while operating between the threshold watermarks. The problem of choosing a good max_p value is that it is highly dependent on the number of flows sharing the bandwidth of the router. It has been shown that the optimal max_p setting increases as the number of simultaneous flows on the router increases [21], thus making it a difficult parameter to choose. A small max_p value may cause the RED algorithm to deteriorate to a tail-drop algorithm where the majority of drops occur when the weighted average queue size reaches max_{th}, on the other hand too high a value may force the average queue size to be lower then the intended or even cause underutilization of the link. The general guidelines [25] for configuring RED suggests a value of 0.1.

Parameter dependence

Configuring the parameters of the RED algorithm is made difficult by the dependency between parameters. All the parameters effects the drop probability function p_b in some way. This is illustrated in Figure 2.4 which shows the drop probability p_b as a function of avg. It is clear that changing any of the parameters max_p, min_{th}, and max_{th} will effect the drop probability described by the function p_b. The parameter w_q, not shown, implicitly affects p_b by having a significant impact on the value of the weighted average queue size.

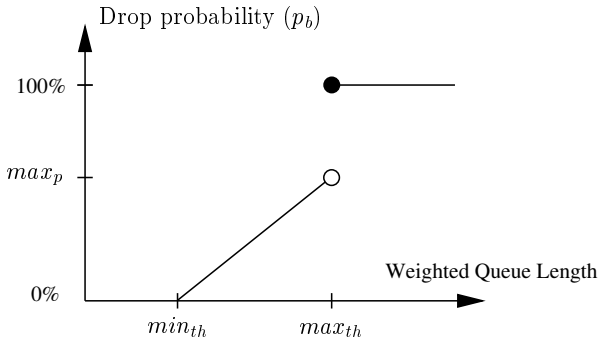


Figure 2.4: *RED drop probability dependence on parameters.*

2.1.4 Aggressive flows

Fairness among flows is an important issue of the Internet. Floyd and Jacobson argue that RED increases fairness among flows compared to tail-drop queuing, by removing the bias against flows that transmit data in bursts. However, fairness among flows in strict sense it supported neither by tail-drop nor RED because this requires per flow scheduling such as Fair Queuing [16].

One problem is fairness among TCP flows that all implement the TCP congestion control algorithms correctly. Another problem is fairness among flows when some of the flows do not react to congestion in a TCP conformant manner. A conformant flow is responsive to congestion notification, and uses in steady-state no more bandwidth than a conformant TCP flow running under comparable conditions [9].

Applications that typically produce non-conformant traffic are multimedia applications. The flows that carry sound or video have requirements different from traditional TCP flows. That is, they may react to congestion, but react differently from traditional TCP flows [54]. Consequently these flows are not conformant in the strict sense and thus may impact the fairness between flows. Because this segment of traffic is growing, it is important to address the fairness issues before it becomes a dominant problem. Another type of non-conformant traffic is TCP implementations that react more aggressively to congestion than what is defined in the TCP standard [2]. For instance an aggressive TCP implementation ignore congestion signals by refusing to reduce the transmission rate.

Traffic not responding to congestion signals is considered a real threat to the Internet, because these flows can cause a significant reduction in the performance of conformant traffic. The problem is that tail-drop or RED queue management does not have a mechanism for isolating conformant from non-conformant traffic. Thus, if a router becomes congested, packets are dropped from both conformant and non-conformant traffic. The conformant traffic will reduce the transmission rate, while the non-conformant traffic continues without changing the transmission rate. This causes the non-conformant traffic to use more than its fair share of bandwidth.

Much research has gone into developing mechanisms that protect TCP conformant traffic from non-conformant traffic. The paper [39] describes flow random early drop (FRED). FRED is derived from RED but has limits on the minimum and maximum number of packets each flow can have in the queue. Furthermore, it has a variable for each flow that counts the number of times the flow has failed to respond to congestion notification, such that these flows can be penalized. Similar mechanisms are described in [3, 53].

Parris [54] has developed the "class based thresholds" (CBT) mechanism. It is also derived from RED. Instead of operating with a single pair of thresholds, CBT has a separate set of thresholds for non-conformant traffic. These thresholds are typically set lower than the thresholds for TCP conformant traffic, thus only allowing the non-conformant traffic to use some smaller amounts of buffer space in the queue. This mechanism differs from FRED because it expects the non-conformant traffic to be identifiable either by a marker in the packet header or simply by the protocol.

Another approach is to use scheduling algorithms. CBQ [31] divides traffic into different classes, and applies scheduling to ensure fairness among these classes.

2.1.5 Summary

In summary the primary goal of RED is to serve as a congestion avoidance mechanism that can increase the overall performance of the Internet. The increased performance is gained by increasing the queueing mechanism's ability to absorb bursts of traffic thus avoiding packet drops, and by improving interactive service by reducing the average packet delay on routers. Additionally, RED will increase the fairness amongst TCP conformant flows by removing the traditional bias against bursty traffic, and by signalling congestion to flows proportionally to the share of bandwidth used by the flow. Finally, RED avoids global synchronization, by avoiding longer periods of congestion signalling.

RED does not provide isolation between conformant and non-conformant traffic. However other queuing mechanisms such as FRED and CBT, have proved to be

effective solutions to this problem.

In general RED adds additional complexity to Internet routers. This manifests itself in the addition of several new parameters for controlling the effect of the algorithm. Even though guidelines exists for parameter tuning [25], the general understanding of the impact of the parameter settings in an environment with Web traffic is limited.

2.2 Related Work

Evaluating the effect of using a queueing mechanism is difficult due to the dynamic and complex environment under which it operates. There are two issues to this: First, does the evaluation model describe an environment well enough to provide sufficiently accurate results? Second, is the described environment a relevant environment?

In the following sections we look closer at how queue management mechanisms have been evaluated, describing the approach used, and emphasizing interesting results regarding RED, and particularly RED parameter settings.

2.2.1 The Original Evaluation

The original RED paper [29] demonstrates that RED operates as intended through four simulations. Each simulation addresses some particular aspect of the functionality that the authors want to examine. In the following we describe these simulations. This is mainly interesting because the method used for the experiments has been one of the main sources of inspiration when designing tests of other queueing mechanisms.

The REAL simulator [37] is used for all the simulations. REAL has later become the basis of the Network Simulator (ns) [46] which is currently the most widely used simulator for testing and experimenting with networks. Each simulation runs for 1-10 seconds, under which the queue size on the router is measured along with packet statistics such as transmission, drop and retransmission times.

In total we present four of the simulations presented in [29]. These can be summarized as follows:

1. Show that RED is able to control the average queue size in response to a dynamically changing load,
2. demonstrate that RED has superior performance over tail-drop queuing when two identical flows are competing for bandwidth,
3. study the effect of using RED in an environment with heavy congestion, and
4. study how RED treats bursty traffic sources compared to tail-drop queuing.

The goal with their first simulation is to show that RED is able to control the average queue size at the router in response to a dynamically changing load. This is done by simulating four FTP flows with different bandwidth-delay products sending traffic, to a sink placed on a shared bottleneck link, through a RED router. The topology of the network is shown in Figure 2.5. Flows are started sequentially with a fixed interval of 0.2s; resulting in a simulation that displays the behavior of RED as the load increases on the router. As expected, the results show that RED is successful in controlling the average queue size on the bottleneck router in response to the dynamically increasing load.

A second simulation demonstrates that RED has superior performance over tail-drop queueing when two identical flows are competing for bandwidth. The topology

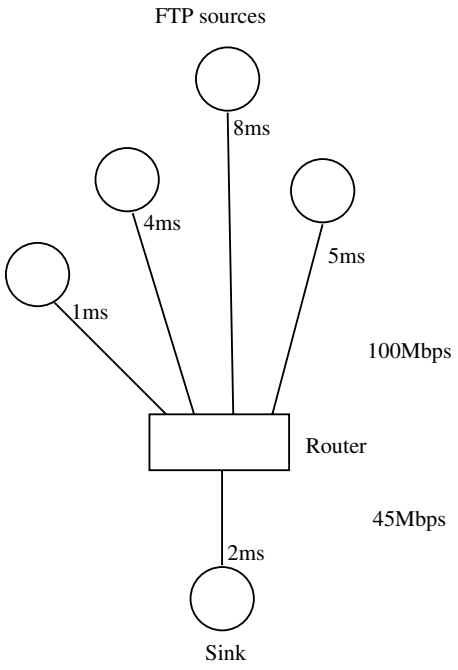


Figure 2.5: *Simulation network for the average queue size test: Four FTP sources each on a separate 100Mbps link and with different round trip times send data to a sink through a router. The link from the router to the sink is a 45Mbps link with a 2ms delay.*

of the simulation network is shown in Figure 2.6. The traffic is produced by two identical FTP flows, being transferred on separate 100Mbps, 1ms delay links to a router. The sink receiving the data from the sources is connected to the router through a 45Mbps, 20ms delay bottleneck link. The two FTP sources are started at slightly different times.

The results show that for TCP flows with large transmission windows, the ratio of throughput to delay is higher with RED than with tail-drop queuing. The reason for this is found to be that tail-drop has to operate with a small queue to ensure low average delay of packets. This has the effect of causing packet drops while the flows are in slow-start, thus reducing throughput. Furthermore, tail-drop is more likely to drop packets simultaneously potentially causing global synchronization effects and loss of throughput.

The third simulation studies the effect of using RED in an environment with heavy congestion. The traffic is the result of having many FTP and TELNET flows, each with a small window and a limited amount of data to send. It should be noted that the authors are not claiming that the traffic in the simulation is a realistic traffic model; they are simply studying RED in a range of environments.

Figure 2.7 shows the topology of the network used in this simulation. It is somewhat more complex than in the previous simulations. The network allows congestion to occur in both directions. This is important because this allows for ack- compression [61, 71] to occur. Ack-compression causes traffic to become more bursty. The reason is that each acknowledgement received by the sender allows it to send another packet. This would normally cause the sender to transmit data at an even rate as the acknowledgements arrive. However, if buffering of packets occur on the path from the receiver to the sender, the acknowledgements are queued in a

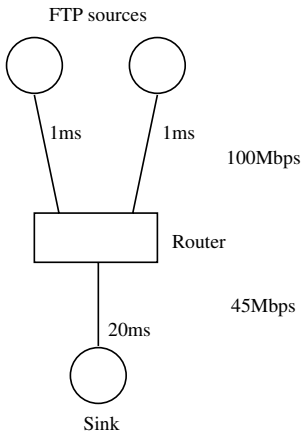


Figure 2.6: *Simulation network for the RED and tail-drop comparison: Two FTP sources with equal round trip times, but on separate 100Mbps links send data to a sink through a router. The link from the router to the sink is a 45Mbps link with a 20ms delay.*

buffer. This can potentially cause the acknowledgements to be compressed because they are forwarded from the router with a shorter interval than when they arrived at the router. Thus, when a sender receives a series of compressed acknowledgements the use of the sliding windows protocol, allows it to send a burst of packets. However, a recent study suggests that ack-compression is in fact a quite rare event [56].

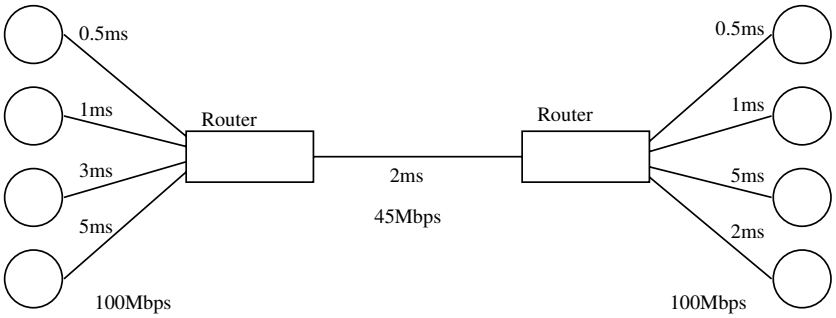


Figure 2.7: *Simulation network for studying effects of heavy congestion.*

In this simulation it is demonstrated that RED is able to control the average queue size with bursty traffic, caused by ack-compression.

The fourth and final simulation in [29] studies how RED treats bursty traffic sources compared to tail-drop queuing. The simulations show that RED does not have the bias against bursty traffic which tail-drop queue management has.

The topology of the network used in these simulations is shown in Figure 2.8. The topology in this simulation is quite similar to the one in Figure 2.5. However, in terms of bandwidth and delay there are significant differences. Five traffic generators send traffic through a bottleneck router to the traffic sink. Four flows, 1-4, have the same round-trip times of 1ms and equal maximum window sizes of 12 packets. The fifth flow is the bursty flow with a longer round trip time of 16ms and a smaller window of only 8 packets. The small window size and larger delay forces

the flow to send packets in bursts because it has to wait for the acknowledgements before sending more packets.

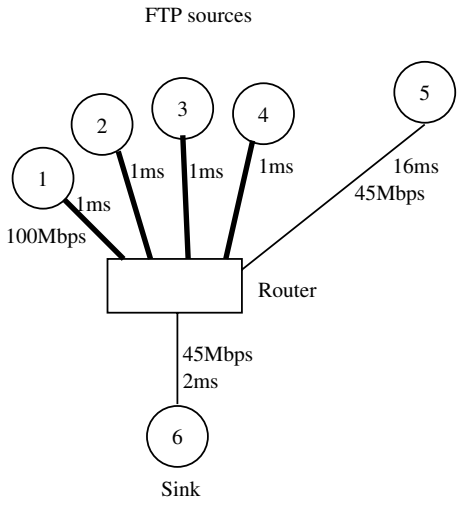


Figure 2.8: *Network topology used for fairness testing.*

The central question studied in this simulation is to see how much bandwidth the bursty traffic source is allowed to use as a function of the amount of bufferspace allocated on the bottleneck router. To study this a series of simulations with tail-drop queue management and with RED was run. In each of these series the buffer size on the bottleneck router was increased.

These results shows that with RED, the fifth flow is allowed to send at its maximum possible transmission rate independent of the the threshold values of the RED queuing mechanism. With tail-drop the buffer has to be quite large before the fifth flow is allowed to send at its maximum transmission rate. The reason is that the fifth flow receives a large percentage of the packet drops on the routers and only a small amount of bandwidth.

Floyd and Fall [26] has later repeated some of these simulations using ns [46] with similar results. Furthermore, Floyd also included random drop [42] in the simulation testing for bias against bursty traffic. The results of these experiments show that routers using random drop also has a bias against bursty sources, and thus perform similarly to tail-drop.

In general, these simulations show that RED works as expected in some specific scenarios. Furthermore, the simulations has supported the choice of recommended parameter settings for RED. However, providing little evidence to the impact RED will have in a setting with a more realistic traffic load.

2.2.2 An Empirical Evaluation

Villamizar and Song report experience with RED in a series of experiments on a Wide-Area network [66]. The overall goal with the study is to demonstrate effects of queuing capacity, and to quantify queuing capacity requirements. As an additional aspect they have included the use of RED in the study.

In general the study shows the importance of providing adequate buffer space and large transmission windows in high speed networks. With regard to RED, the study shows that RED slightly increases the overall throughput in the network. However more importantly, it was shown that RED can be used for removing po-

tential congestion collapse due to oversized transmission windows, that exceed the storage capacity of the network.

The test network used was an unchannelized DS3 network consisting of FDDI rings and DS3 circuits that resemble the equipment used in NSFNET and ANSNET. Two paths through the network are used. The first runs on a fairly direct path between New York and Michigan providing a WAN with a 20ms round trip time. The second path also runs between New York and Michigan, but via Texas,with a total round trip time of 68ms.

The primary bottleneck on the paths is the first DS3 circuit. Experiments are done with two different buffer sizes and with both tail-drop and RED queue management.

Traffic in these experiments consists of 1, 4, or 8 flows sharing the bandwidth. The assumption is that these cases are likely to be far more bursty than aggregations of very large numbers of slower TCP flows. Furthermore, these cases constitute worst case situations which may arise if supercomputer centers are encouraged to make use of high bandwidth applications.

The flows are tuned to provide the highest possible throughput by allowing a maximum transmission window size up to 512KB. Furthermore, a series of experiments were done with a reverse traffic flow. This forces ack-compression of the forward data flow, and thus more bursty traffic, because the reverse traffic flow causes packets to be queued at routers.

A very important result in this paper is that RED seems to be able to remove the performance tuning dilemma of choosing a good maximum transmission window size. The problem is that to achieve maximum throughput for a given flow, one must choose a window size large enough to keep the flow active, and small enough to avoid injecting too many packet into the network. More precisely, the window size should be equal to the bandwidth-delay product or equal to the queuing capacity of the network. If the window becomes too large, congestion will occur, and as demonstrated in the paper, this can have a very significant impact on the performance of high bandwidth flows.

The reason for the severe performance degradation seen in the paper is that congestion in a network with traditional tail-drop queuing will cause longer sequences of packet drops, e.g. sequences where dropped packets are not retransmitted before the next drop occurs. This causes TCP to recover using retransmission timeout instead of fast retransmit. However, it is found that when using RED, congestion is signalled in such a manner that flows generally recover from congestion using fast retransmit. Thus, the danger of choosing too large transmission windows is removed.

Another important result is that if queuing capacity is inadequate at some bottleneck, then this will limit the bandwidth utilization. At higher loads they found a greater tendency toward degradation or what is referred to as a mild congestion collapse. In general it was found that routers should have queueing capacity of more than 1-2 times the bandwidth-delay product. RED was found to be helpful for reducing the queue utilization, especially for a larger number of flows, but could not compensate for inadequate queuing capacity.

2.2.3 Alternative Queuing Mechanisms

Several alternatives to the RED mechanism have been suggested in the literature: flow random early drop (FRED) [39], Balanced RED (BRED) [3], BLUE [20], Stabilized RED (SRED) [53], and Adaptive RED [21]. Some of these focus on issues related to aggressive flows, while others suggest optimizations or alternative approaches to handling the congestion avoidance problem. Our interest here is not on

these specific algorithms, but on how they have been evaluated and whether any specific results regarding RED have been reached.

FRED

The motivation for developing flow based RED (FRED) [39] is found in a series of simulations that display down sides of the RED mechanism using simulations that closely resemble those from the original RED paper, see Section 2.2.1.

The first simulation use a network topology similar to the topology used in [29] for studying REDs ability to avoid bias against a bursty traffic source, see Figure 2.8. Four high bandwidth and low delay flows share a bottleneck link with a fifth and fragile flow that has a higher per packet delay and less bandwidth. Measurements taken during these simulations show that RED managed to keep the packet drop distribution proportional to the bandwidth distribution, but that the bandwidth capacity of the bottleneck link is not shared fairly among the flows. Therefore, dropping packets proportional to the resource consumption does not lead to fair sharing of bandwidth.

The second simulation studies how the fragile flow is punished when competing with four non-fragile flows. This is done by first letting the fragile flow run without the competing traffic, showing that the fragile flow can consume up to 39% of the bottleneck bandwidth. Repeating the simulation again, now with the competing traffic, shows that the fragile flow is unable to obtain its 20% share of the bottleneck bandwidth.

In total, these simulations demonstrate that RED does not provide fairness among competing flows, even though packets are being dropped proportional to the resource usage.

BLUE and Adaptive RED

BLUE [20] and adaptive RED [21] are both suggestions for replacements of RED for congestion avoidance. Both mechanisms are suggested by the same authors.

The BLUE mechanism [20] is compared with RED through a number of simulations where the RED weight parameter w_q is evaluated. The particular interesting aspect is that the traffic in this study consists of 1000 to 4000 traffic sources that generate traffic with Pareto on/off periods[1]. Consequently this might provide clues to the behavior in an environment with web like traffic. However, since the test uses Explicit Congestion Notification (ECN) [24] where packets are marked instead of dropped, this study is not directly comparable with evaluations using packet drops.

Simulations in [21] studies the impact of the maximum drop probability max_p of RED on the drop rate compared to tail-drop queuing. The simulation setup is again that a number of traffic sources send traffic to a sink through a bottleneck router. The traffic consists of either 32 or 64 concurrent TCP flows sending to the sink.

The simulations lead to the observation that RED has marginal impact on the number of packet drops. By fine tuning the max_p value, the drop rate can be reduced with up to 0.5% compared to tail-drop queuing. However, choosing too small a value leads to loss rates close to those experienced with tail-drop, and choosing larger values of max_p leads to a increase of up to 2% in the packet loss rate.

Additionally, the simulations show that the optimal value of max_p is a function of the number of flows carried by the router. This means that for the optimal max_p value for 32 flows was slightly smaller than the optimal value for 64 flows. The optimal max_p values varies around 1/10.

[1] The Parato distribution is a heavy-tailed distribution, see also Section 3.2 on page 26.

A general argument is made that the effectiveness of RED decreases as the number of flows sharing the queue increases. This is because the proportion of flows that actually receive and act on RED-induced congestion indications is reduced, as the number of flows sharing the queue increases.

Stabilized RED (SRED)

SRED [53] is compared to the performance of RED through two main simulations. The first simulation studies buffer occupancy with 10-1000 persistent flows sending through a 45Mbps bottleneck link. The key observation from this simulation is that the buffer occupancy of RED increases with the number of flows sharing the link. This means that with a small number of flows the buffer utilization is at or below minimum threshold, while with a large number of flows, it stabilizes around the maximum threshold.

Another set of simulations use a more realistic traffic model to compare the performance of SRED and RED. The traffic model used here is derived by Neidhardt [50], and is based on the data generated by Web access of students at Boston University [12]. The model consists of two distributions: a file size distribution that describes the file sizes of retrieved Web objects, and a think time distribution that describes the time between transfers. Between each transfer the flow is forced into slow start thus modeling the behavior of HTTP 1.0, where each Web object is transferred by a separate flow. To increase the traffic load produced by the model the think times are divided by 10.

From the simulations with the more realistic traffic model only measurements of buffer occupancy is reported. These measurements show that buffer occupation tends to stabilize around the maximum threshold for larger number of active flows.

2.2.4 Analytical Evaluation

There are only a few evaluations that solely focus on evaluating the impact of using RED. However, Martin May *et al.* has published two: an analytic and an empirical evaluation. We first look closer at the analytic evaluation [45] and then the empirical [44].

The analytical evaluation compares the performance between RED and tail-drop queueing. The analysis quantifies how RED influences the packet loss rate under traffic conditions with mixes of bursty (TCP) and smooth (UDP) traffic. The focus is especially on patterns of consecutive loss, mean delay, and delay jitter of packets.

The results from their analysis was validated through simulation using ns [46]. The network used in the simulation consists of a bottleneck router that feeds a 10Mbps link, receiving traffic from several 100Mbps links. The traffic consists of over 10-300 simultaneous flows. To introduce burstiness in the traffic the different flows experience different round trip times ranging from 120ms to 220ms. UDP traffic is also generated. This traffic is sent at a constant bit rate, consuming 10% of the bottleneck bandwidth. During the simulations the authors measure drop rates of both TCP and UDP traffic and the delay of the UDP packets. Furthermore, the authors calculated the "goodput" for the TCP flows. The goodput of a flow is the bandwidth delivered to the receiver, excluding duplicate packets [27].

Overall, the results show that RED does not improve the TCP "goodput" significantly, and that this effect is largely independent of the number of flows. Furthermore, the analysis shows that RED has a lower mean queuing delay but increases delay variance.

The empirical evaluation directly addresses the problem of tuning RED parameters. This is done empirically using the Cisco WRED [11] implementation on a Cisco router, where WRED is Cisco's own implementation of RED. As with the

analytical approach the traffic consists of a mix between TCP and UDP traffic: 80% TCP traffic and 20% UDP traffic. 60% of the TCP traffic is FTP like traffic with an average file size of 100KB, and the remaining 40% is HTTP like traffic with a mix of small and large file sizes. The key performance indicators are throughput, bytes sent, and the loss rate for UDP traffic.

The results show that WRED parameters generally have little impact on the traffic, and that it is difficult to choose good values for the parameters. In comparison with tail-drop queuing, the study shows that the Cisco implementation of RED does not exhibit much better performance than tail-drop queuing. Only in situations where the amount of bufferspace is increased does RED seem to give some improvement.

Both of these evaluations provide interesting results, however none of them link the packet loss rates and delay to the end-to-end response times for interactive web-like traffic. Furthermore, the empirical evaluation is not directly comparable with experiments with RED because the Cisco WRED implementation is used.

2.2.5 Feedback Control

Firoiu *et al.* [23] model RED as a feedback control system which allows the authors to discover fundamental laws governing the traffic dynamics in TCP/IP networks. Based on these, the authors derive a set of recommendations for the the architecture and implementation of congestion control modules in routers. The results of the analysis are validated through simulation with ns [46].

The traffic assumed in the model is a constant number of concurrent flows. Furthermore the model assumes that all flows have the same fixed average round trip time.

An interesting observation with regard to RED parameter setting is their discussion covering the weight, w_q, used in the weighted average queue size calculation of RED. The authors see two opposing conditions for choosing a good w_q value. The first condition is that w_q should not be influenced by the linear increase and multiplicative decrease of transmission rates of the flows. The second condition is that the queueing algorithm should change its weighted average queue size as fast as possible to match change in traffic conditions. This includes changes in the number of flows or round trip times.

The general recommendation is that w_q should be set equal to the periodicity of the TCP flows. Where the periodicity is the period between experiencing congestion, decreasing the sending rate, and increasing it again until congestion is detected.

2.2.6 Pilot Tests

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 [17] and Reynolds at QualNet [59] (now Verio). Doran's measurements using the Cisco implementation indicate that RED was able to sustain near 100% utilization on a 1920Kbps customer-access link which tail-drop queuing 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. Overall the performance was somewhat superior to tail-drop queuing. The default values for the maximum drop probability of 1/10 and the weight of 1/500 was used and their effects not studied.

For customer access DS1 links, (apparently) the default settings were used. These 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..." [59].

2.2.7 Summary

In the previous section we have described the methods and in some cases results of evaluations queue management algorithms described in the literature. Overall we can group the described studies in two categories: studies of specific behavior in a simple environment with only a few competing flows. This category includes the original studies of RED described in Section 2.2.1 and the experiments with FRED described in Section 2.2.3. The second category includes studies that focus on providing a realistic scenario under which an active queuing mechanism can be tested. In this category we have evaluations such as the one of BLUE, and of SRED in Section 2.2.3 and the analytical evaluations that we described in Section 2.2.4.

With our goal of providing an realistic environment for studying the effects of using RED the category of evaluations with a realistic environment is of central interest to us. However, several of the studies have significant differences in terms of environment making i impossible to compare results. For example in one case a different congestion notification mechanism is used and in other cases there are algorithmic differences between the evaluated queuing mechanism and RED. Furthermore, while these studies provide a realistic scenario the evaluations mainly focus on network central measures such as drop-rate, throughput and goodput and not on the impact on the end-to-end performance.

2.3 Summary

In the previous sections we have first given a detailed description of the RED queueing mechanism. We have then studied different approaches for evaluating the effects of active queueing mechanisms, and especially focused on any results regarding REDs impact on the performance of Web like traffic. However, only a limited amount of information is available.

None of the evaluations of RED focus on the impact that RED may have on Web traffic response times. Response time is the central parameter for the end user when browsing Web pages on the Internet. Therefore, we believe that it is important to study and understand the effect that RED may have on the performance of Web response times.

Chapter 3

Traffic Generation

In this chapter we focus on describing how traffic is generated for our study of RED queue management. Generating a realistic traffic load is not a new concept and there has been conducted considerable research on the topic. A central motivation is to provide better modelts for performance simulation of algorithms such as flow control and congestion control algorithms.

In the following we first look at the background of traffic modeling. We then look closer that the general properties observed in studies of Web traffic. Based on this we describe the traffic generation tool used for generating Web-like traffic on the laboratory network.

3.1 Related Work

Network traffic is one of the basic elements necessary when performing network experiments or simulations. In some cases it is sufficient to simply have one or two competing connections or some other simple scheme to prove a specific point. An example of this is the original evaluation of RED described in Section 2.2.1 on page 15, where a very simple traffic load is used for illustrating a specific aspect in an evaluation. However, in our case it is a central point to have a realistic traffic workload.

Some of the earliest work on generating a realistic workload in Internet network simulations was performed by Danzig and Jamin in their work on TCPlib [14]. TCPlib is a library that includes empirical models describing characteristics of the most popular applications at that time, such as TELNET, NNTP, and SMTP.

The models used in TCPlib, see [10, 15], are derived from network packet traces. Each application is described through a number of cumulative distribution functions describing the probability distribution of characteristic events. These distributions are used for modeling a particular application in a simulator. The simulator follows the flow of the modeled application, and when necessary, it makes choices by random sampling from the distributions describing the characteristic events.

For instance, the model describing the behavior of TELNET conversations consists of three distributions: the duration of the conversations, the interarrival times of packets, and the distribution of packet sizes. An instance of a TELNET connection is modeled by letting the simulation follow the behavior described by the flow chart shown in Figure 3.1. Each time a duration, interarrival time, or packet size is needed it is simply random sampled from one of the distributions describing the model.

A key aspect of this method for modeling traffic is that the model only describes the application behavior. This means that low level protocol behavior is abstracted

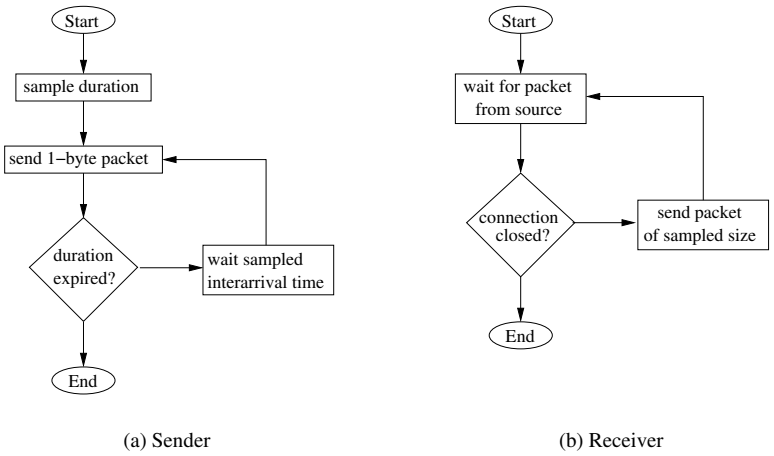


Figure 3.1: *Flow chart of a TELNET connection.*

out of the models, and thus the models are independent of both low level protocol behavior and specific network characteristics.

Danzig and Jamin used only empirically derived distributions in their library. Another approach is to derive analytical functions from the empirical distributions. Paxson describes how he derived analytical models of Wide-Area TCP connections in [55]. He explains how this has several advances compared to empirical distributions. The analytical distributions are easier to convey and analyze, because they have more compact and descriptive representations. For instance, by making an analytical model we know which group of distributions, normal, exponential, or other, that can be used for describing an empirical distribution. This gives general insight on the behavior that can be observed when using the model.

Paxson concludes that both empirical and analytical models can be used for modeling wide-area network connections. Though pointing out that for both empirical and analytical models, we must be careful because of frequent discrepancies of the upper 1% tails. Furthermore, the work shows that the significant variation in network traffic means that both types of models must be a somewhat rough compromise.

3.2 Web Traffic

A number of studies of network traffic behavior provides important results that should be taken into account when generating Web traffic, thus ensuring that the properties of real traffic remain present in the generated traffic. In the following sections we look closer at the statistical properties of Web traffic and how we can ensure that these properties remain present when generating Web traffic.

In 1994 Leland *et al.* [38] published a study of the statistical properties of local area Ethernet traffic. By analyzing per packet network traces the authors found evidence that the traffic was *self-similar* while also being extremely bursty, and even being bursty independent of the timescale at which the traffic is viewed. This lead to the hypothesis that network traffic, in addition to being self-similar, has *long-range dependence*. This essentially means that the models used at that time did not correspond well to the actual behavior observed in actual traces of network traffic.

A later study of Web traffic [13] by Crovella and Bestavros shows evidence of

presence of long-range dependence in Web traffic, thus providing evidence that the hypothesis from [38] holds for Web traffic. Crovella and Bestavros used packet network traces collected from their local site, as it was done in [38]. In addition, they were able to collect very exact data describing Web object sizes, transfer times, and other Web activity by instrumenting the Web browsers used on their site. By analyzing this data they were able to provide evidence that Web object sizes and quite periods are *heavy tailed* and thus the presence of long-range dependence.

The consequence of these results means that we can describe the characteristics of Web traffic parsimoniously using a simple statistical model, called the ON/OFF model or packet train model in combination with heavy-tailed distributions. The advantage of this is that this model is well understood and only depends on a few parameters, making it simple and less sensitive to errors, thus ensuring that we are on the right track when modeling Web traffic.

In the following sections we first describe the central concepts: self-similarity, heavy-tailed distributions, and long-range dependence. Then we explain how the ON/OFF model can be used for modeling Web traffic.

3.2.1 Self-similarity

Self-similarity is a property that seems often to arise when analyzing network traffic [38, 57, 13, 18]. As we shall see the general strength of showing that measured data is self-similar lies in the fact that there is a repetitive pattern in the data. Consequently, it is only necessary to study a limited set of data in order precisely describe the behavior or network traffic. Furthermore, the repetitive pattern makes is easy to reproduce self-similar traffic with a traffic generator.

In the following we first describe self-similarity on a set and show how self-similarity can be used to calculate the dimension of the set. We then describe probabilistic self-similarity which is the type of self-similarity detected in measurements of network traffic.

To illustrate the principle of self-similarity, we describe how to calculate the fractal dimension of the von Koch curve.

Example

To calculate the dimension of a set S we cover the set with a minimal number of spheres N_r, where each sphere has a fixed diameter $r \neq 0$. Now assume that we have $N_r r^{d_r} = 1$, then:

$$r^{d_r} = \frac{1}{N_r} \Leftrightarrow d_r \cdot \ln(r) = -\ln(N_r) \Leftrightarrow d_r = -\frac{\ln(N_r)}{\ln(r)} \tag{3.1}$$

where d_r is the integer dimension of the set S. In most cases the dimension is an integer value, however in some special cases it can also be a non-integer. To handle this case we extend the notion of dimension by computing the limit as $r \to 0^+$:

$$d = \lim_{r \to 0^+} d_r \tag{3.2}$$

we call d the fractal dimension.

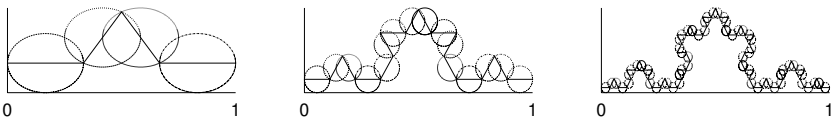
An example of a set which has non-integer dimension is the von Koch curve, which can be described by the following recursive algorithm:

1. Start with a line segment. Divide it into thirds. Place the vertex of an equilateral triangle in the middle third.

2. Copy the whole curve and reduce it to 1/3 its original size. Place these reduced curves in place of the sides of the previous curve.

(a) order = 1 (b) order = 2 (c) order = 3 (d) order = 5

Figure 3.2: *von Koch curves from 1 to 5 iterations.*



(a) order = 1, 4 spheres (b) order = 2, 16 spheres (c) order = 3, 64 spheres

Figure 3.3: *Covering von Koch curves with spheres.*

3. Return to step 2 and repeat.

Figure 3.2 shows the von Koch curve at 1,2, 3, and 5 iterations of the algorithm. To calculate the dimension of the von Koch curve we cover the curve with spheres each with a fixed diameter, as we described earlier. Figure 3.3, shows the result of this process for the first few iterations. Based on this we can see that in this case the number of spheres and the diameter is a function of the number of iterations: $N_r = 4^n$, and $r = \frac{1}{3^n}$ where n is the number of iterations. Using Equation 3.1, we can calculate the dimension d_n:

$$d_n = -\frac{\ln(4^n)}{\ln(\frac{1}{3^n})} = \frac{\ln(4)}{\ln(3)}, \quad \forall n \geq 1 \tag{3.3}$$

Using this we determine the fractal dimension of the von Koch curve:

$$d = \lim_{r \to 0^+} d_r = \lim_{n \to +\infty} d_n = \frac{\ln(4)}{\ln(3)}, \text{ and therefore } 1 < d < 2 \tag{3.4}$$

thus we have showed that the von Koch curve has non-integer dimension. To do this we have used self-similarity:

Exact self-similarity Let S be a set. S is said to be self-similar if for any open subset $S' \subset S$, there exists a bijection $h : S' \to S$ such that $h(S') = S$.

In the context of the von Koch curve, self-similarity describes the infinitely repeating repetition of the curve we used for generalizing the calculation of N_r and r. More generally self-similarity can be described as the "zoom effect" where the exact same pattern will repeat itself infinitely independent of the zoom level.

Probabilistic self-similarity

Let Y be a stochastic continuous-time process $Y = \{Y(t), t \geq 0\}$, with continuous time parameter t. Y is called self-similar with self-similarity parameter $0 < H < 1$ (Hurst parameter), if for any positive stretching factor c, it satisfies:

$$Y(t) \stackrel{\mathrm{d}}{=} c^{-H} Y(ct), \quad \forall t \geq 0 \tag{3.5}$$

If the rescaled process $c^{-H}Y(ct)$, with timescale ct, is equal in distribution to the original process $Y(t)$, then $Y(t)$ is self-similar. As with the deterministic self-similar process we can "zoom" in on a probabilistic self-similar process. However, instead of seeing the exact same picture, we will see that each zoom-level will look qualitatively the same, see [6, page 48].

The classical example of a stochastic process that has been proved probabilistic self-similar is fractional Brownian motion (fBm) with $1/2 < H < 1$ [69]. Fractional Brownian motion is special process because it is both self-similar and has a non-integer dimension, by definition this means that fractional Brownian motion is a fractal.

3.2.2 Long-range Dependence and Heavy-tailed Distributions

Long-range dependence refers to a stochastic process that has long memory, because the dependence between events that are far apart diminishes very slowly with increasing distance [6]. Long memory processes can be described using heavy-tailed distributions that satisfy:

$$P[X > x] \approx x^{-\alpha}, \quad \text{as } x \to \infty, \quad 0 < \alpha < 2 \tag{3.6}$$

Whether a distribution is heavy-tailed depends only on the higher values of the random variable e.g. the tail of the distribution. If the asymptotic shape of the distribution is hyperbolic, it is heavy-tailed. Heavy-tailed distributions and thus long-range dependence can be modeled parsimoniously with Parato distributions. A Parato distribution is hyperbolic over its entire range. With the shape parameter α and the location parameter k it has the cumulative distribution function:

$$F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}, \quad k, \alpha > 0, x \geq k \tag{3.7}$$

with the corresponding probability density function:

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \quad \alpha, k > 0, \ x \geq k \tag{3.8}$$

If $\alpha \leq 2$, then the distribution has infinite variance, and if $\alpha \leq 1$, then it also has infinite mean. The parameter k represents the smallest possible value of the random variable.

3.2.3 ON/OFF model

As we stated earlier, the work by Crovella and Bestavros [13] shows that we can use the ON/OFF model with heavy-tailed distributions, as described in [69], to describe the behavior of Web traffic.

The use of the ON/OFF model is based on the observation, made by Jain and Routhier in [35], that packets in networks travel in trains. Thus the interarrival time between trains from a traffic source is much greater then the interarrival time between packets.

This observation is used in the ON/OFF model, by describing traffic sources as being strictly alternating between being ON (transmitting) or OFF (quite). Hereby assuming that data is being sent as constant bit rate when ON.

ON and OFF periods are controlled by two distributions, one describing the lengths of the ON periods and one describing the lengths of OFF periods. These distributions are required to be independent and identically distributed (i.i.d.), that is the distribution describing the length of ON times must be independent of the distribution describing OFF times, and vice versa. Additionally, identically distributed requires that it does not change over time.

Mandelbrot originally developed and used the ON/OFF model [41, 69]. He showed that the aggregate load produced from a large number of ON/OFF sources that are i.i.d. is self-similar. However, as described in [69], if the ON and OFF distributions have finite variance, the aggregate load will be similar to white noise, which is significantly smoother than the traffic found in network traces [38]. Therefore we need to use a more restrictive version of the ON/OFF model which was developed by Taqqu and Levy [64].

Taqqu's and Levy's more restrictive model has the additional requirement that either the ON or OFF distribution is a heavy-tailed distribution with infinite variance, such as a Parato distribution with $\alpha < 2$. This has significant influence on the traffic generated by the model. In fact, it can be proved that with a large number of sources the aggregate traffic from this model is approximately equivalent with the behavior of a fractional Brownian motion with dimension $1/2 \geq H < 1$ [68].

This result is important because it also explains why self-similar behavior can be observed in network traffic. Leland *et al.* found evidence of self similar behavior in Ethernet traffic. Crovella and Bestavros found evidence showing that the distribution of ON and OFF times are heavy-tailed. Taqqu' and Levy's proof that the aggregate traffic from a large number of sources is approximately equivalent with the behavior of a fractional Brownian motion links heavy-tails and self-similar behavior together because we know that a fractional Brownian motion is per definition self-similar.

A more general consequence of this result is that we can use the ON/OFF model for modeling Web traffic. In fact this model gives us a very simple and exact method of modeling traffic. The only really important requirement is the presence of a heavy-tailed distribution. Also, using the ON/OFF model when generating Web-traffic from a traffic model is quite simple since the model already defines the behavior of each individual source.

3.2.4 Summary

Returning to the hypothesis that network traffic, in addition to being self-similar, has long-range dependence, which we described in Section 3.2. Leland *et al.* observed that Ethernet traffic exhibited self-similar behavior, yet the traffic remained bursty over several timescales. This lead to the hypothesis that traffic could be modeled with the restrictive ON/OFF model, which requires the presence of heavy-tailed distributions on the traffic sources.

As described, Crovella and Bestavros [13] continued the work of Leland *et al.*, however focusing solely on analyzing the behavior of Web traffic. In their model of Web traffic, ON-times correspond to transmission times of individual objects, and OFF-times correspond to the intervals between transmissions.

Through an analysis of the ON- and OFF-times they determine that both ON- and OFF-times in their traces are heavy-tailed, and can be well modeled with a Parato distribution. Further analysis of the ON-times shows that the primary reason for ON-times being heavy-tailed is the sizes of the transferred Web objects, because this distribution also turned out to be heavy-tailed. Finally, because both the ON- and the OFF-distribution can be modeled with a heavy-tailed Parato distribution, it is nessesary to compare the α parameters of each of these to determine which of them has the most dominant role in causing the fractal like behavior of Web traffic. The result of this comparisson points to the distribution of ON-times. Modeling ON- and OFF- times with the Parato distribution, ON-time has $\alpha \approx 1.0 - 1.3$ while OFF times has $\alpha \approx 1.5$. Thus they conclude that ON-times is the most dominant factor, since this distribution has the "heaviest tail".

3.3 The Mah Traffic Model

The central element of the Web-like traffic generator is the model from which the traffic generator generates traffic. In our case we have chosen to use an empirical model developed by Bruce Mah [40], that describes the behavior of a user browsing the Internet in terms of the traffic generated by a user.

The model describes the behavior of clients retrieving pages from Web servers using HTTP 1.0 [8]. The model was derived using basically the same methods as those used in TCPlib [14]. This means that the traffic model is an application level model that has been derived from network packet header traces taken from a backbone network. Additionally, Mah has taken into account the analysis described in Section 3.2 by Crovella and Bestavros [13] by testing the distribution of Web objects sizes for being heavy-tailed. In total, making the Mah model well suited for our needs.

3.3.1 Behavior

The behavior of Web traffic in the Mah model is characterised by user behavior, the internal structure of Web pages, and the size of pages being transferred. In the following we describe the distributions and the overall behavior described by the model.

A Web transfer is initiated by the client when a Web page is selected for retrieval from a Web server. A page generally consists of a HTML [7] document with a number of embedded references to images, sounds, or other. Retrieval of the page consists of first retrieving the main HTML document, and then retrieving embedded objects, as the main HTML document is being parsed. When using HTTP 1.0 [8], each object of the page is retrieved on a separate TCP connection and thus browsers tend to use concurrent connections in order to reduce the time it takes to retrieve a page.

Distributions

Mah has chosen a set of six distributions for characterising the general structure of Web pages. These are summarised in Table 3.1. At the lowest level the *request* and *reply length* distributions that characterise the sizes of objects being transferred to and from Web servers. For characterising the structure of pages the distributions *document size* describes the number of objects per Web page. User dependent behavior is characterised through the three distributions: 1) *think time*, 2) *consecutive document retrievals*, and 3) *server selection*. The *think time* distribution describes the time a user spends looking at a retrieved Web page before continuing to the next page. Consecutive document retrievals describes how many pages a user retrieves from a server before moving on to the next server, and server selection describes the relative popularity of the Web servers.

As we shall see in Section 3.3.2, where we describe each distribution in detail, the actual number of distributions is somewhat larger. The reason is that Mah found that request and reply lengths are better modeled when dividing them into two groups. One for primary requests and replies, and one for secondary requests and replies. Where primary refers to the HTML document and secondary refers to any objects embeded in the HTML document.

Flow

Using the approach in [14] we describe the actual flow of the model by a flow diagram. This is shown in Figure 3.4. The diagram shows both the flow of the browsing user and of the server.

Distribution	Units	Description
request length	bytes	HTTP request length
reply length	bytes	HTTP reply length
document size	objects	Number of objects per document
think time	seconds	Interval between retrieval of two successive documents
consecutive document retrievals	pages	Number of consecutive documents retrieved from any given server
server selection	server	Relative popularity of any Web server, used to select each succeeding server accessed

Table 3.1: *Summary of distributions in the HTTP traffic model.*

Following the flow of the browsing user in the flow diagram, we see that browser first selects which Web server to access. It then decides on the number of pages to retrieve from the server, and selects the size of the next document is by first sampling from *consecutive document retrievals* and then from *document size*.

Based on these values the browser connects to a Web server to retrieve the page. Each object in the page is retrieved from the server by first establishing a TCP connection to the chosen server, and then sending a request. The size of the request is selected from *request length*. Once the server receives the entire request from the client it responds by sending a reply with a size sampled from *reply length*.

Once a page has been retrieved we model that the user thinks for a while before proceeding to the next page. This is simply done by waiting for some amount of time sampled from *think time*. After waiting, the model either proceeds by retrieving another page from the same server or starts over, and selects another server, dependent on the value sampled from *consecutive document retrievals*.

The actual behavior of a browsing client implemented in the traffic generator is somewhat more complex then what we just described. The reason is that commercial Web browsers use several concurrent connections to reduce the time it takes to retrieve a Web page with more than one object. To model this we allow up to two simultaneous connections per browser even though some browser products are know to use more [67]. Furthermore the model in the implementation includes steps for terminating after a certain period of time.

3.3.2 Distributions

The distributions used in the Mah model of Web traffic has been derived from network packet header traces. These were collected on the Computer Science Department backbone at University of California at Berkeley in late 1995. Table 3.2 summarises the four traces that were used for the model: the date at which the trace was taken, duration of the trace, number of HTTP packets in the trace, and the number of HTTP request/reply pairs found in each of the traces.

In the following we look closer at the empirical distributions that Mah had derived from the network traces.

Page Length

The distributions describing the *page length* plays a key role for several of the other distributions. The problem is that packet header traces do not exactly describe the number of elements in a page and whether two connections transfer objects

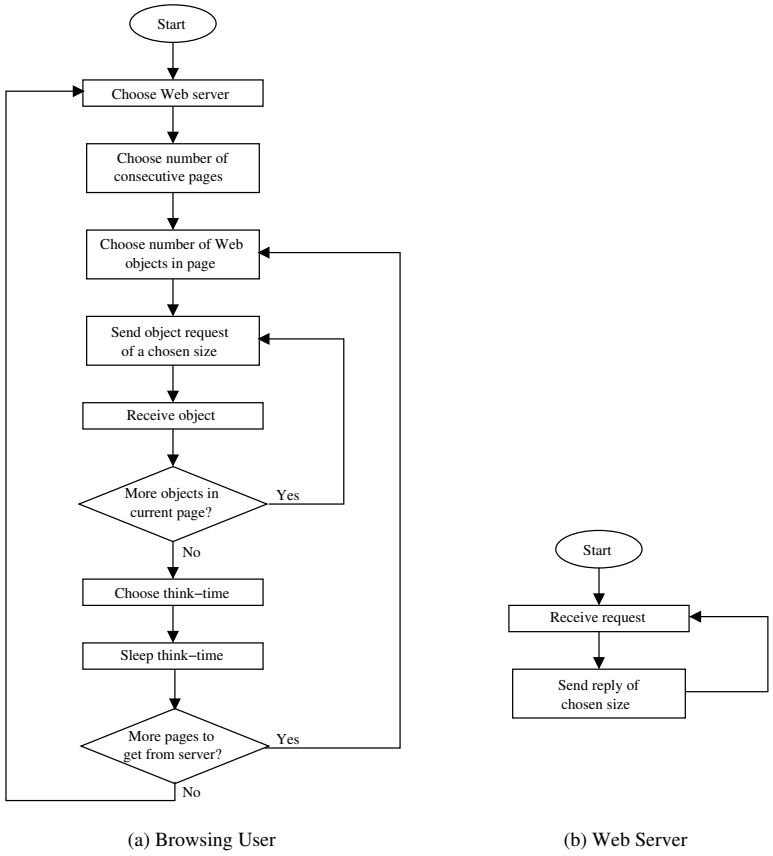


Figure 3.4: *Flow diagram of Mah's HTTP traffic model.*

belonging to the same page. Therefore it is necessary to use a simple heuristic to estimate the approximate length of a page in terms of number of Web objects.

This heuristic says that two connections belong to the same page if the client addresses are identical, and if the time from when the first connection completes until the next connection starts is less than a time interval T_{thresh}. Note that this definition also allows for concurrent connections since this would just give a negative time difference.

By deriving a number of different distributions for the page length with different T_{thresh} values, Mah arrived at choosing to set $T_{thresh} = 1s$. The intuition was that users generally take more than one second to react to the display of a new page and requesting a new document. Furthermore, as we can see from Figure 3.5,

Starting Date	**Duration (hr:min)**	**Packets**	**request/reply pairs**
19 Sep 1995	39:40	186068	5030
11 Oct 1995	29:31	458264	5699
1 Nov 1995	25:30	369671	3659
20 Nov 1995	138:14	676256	18034

Table 3.2: *Summary of traffic traces [40].*

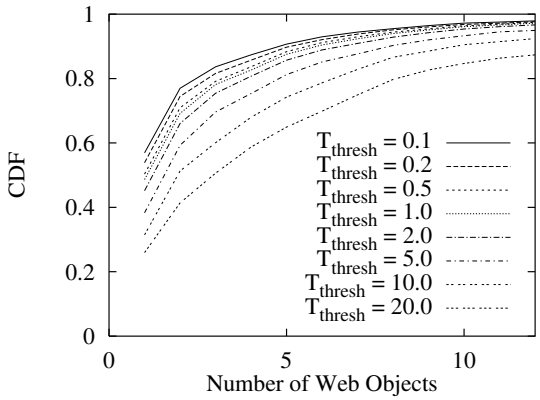


Figure 3.5: *Cummulative Distribution Functions (CDFs) of Document Length in Objects, 19 September 1995. Curves correspond to varying of T_{thresh} in seconds [40].*

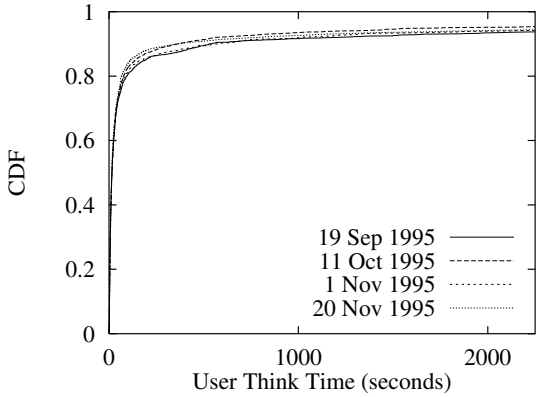


Figure 3.6: *CDF of user think times for all traces [40].*

choosing other values in the same range ($\pm 1s$) provided results close to the one with $T_{thresh} = 1s$.

User Think Time

Closely related to the page length is the *user think time* distribution, since this distribution also depends on the choice of T_{thresh}. Figure 3.6 shows the Cummulative Distribution Function (CDF) for *User think time* from 0 to 2000 seconds and with $T_{thresh} = 1s$. The distribution seems to be consistent with the think time distribution found in [12], however Mah does not test whether these think time distributions are heavy-tailed. Generally, the median think time is about 15s, and 95% of the think times are lower then 30000s. The highest think times are longer then 24hours.

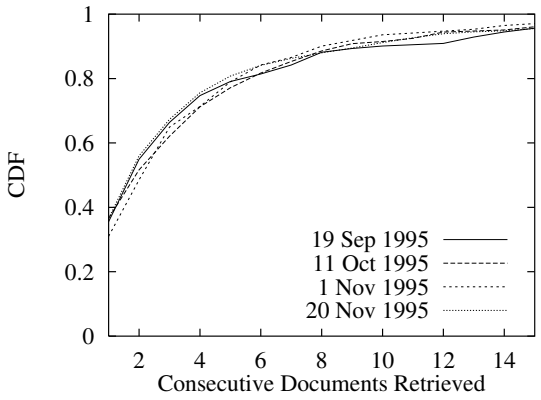


Figure 3.7: *CDFs for consecutive document retrievals for all traces [40].*

Consecutive Document Retrievals

Consecutive document retrievals describes the number of pages that users retrieve from Web servers before proceeding to another server. Figure 3.7 shows the CDF for consecutive document retrievals. As we can the median for consecutive document retrievals lies around two pages before moving on the the next server. However, some rather long sessions where a user retrieves dozens of pages from a single server before moving on were also found. Furthermore we should note that this distribution indirectly depends on the choice of T_{thresh} because it requires the requests to be grouped into pages.

Request Length

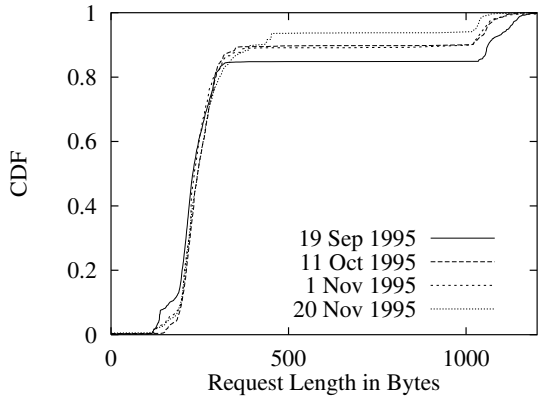
The CDF of the *request length* is shown in Figure 3.8a. As observed by Mah, the request length distribution shows bimodal behavior, by having at least two separate groups of request lengths [40]. The first group, which covers around 80-90% of the requests, uses around 250 bytes per request and the second group uses around 1KB per request. Mah argues that this could be due to different types of requests where the group around 250 bytes corresponds to simple Web object retrievals, while the group around 1KB corresponds to more complex requests such a those generated by HTML forms. However, he is unable to provide proof simply because the packet header traces do not provide this information.

Grouping requests by pages when finding the page length distribution also allows requests to be grouped into primary and secondary requests. Primary requests consists of the requests made for the first Web object of each Web page. Secondary requests consists of all the remaining requests made to retrieve the entire page.

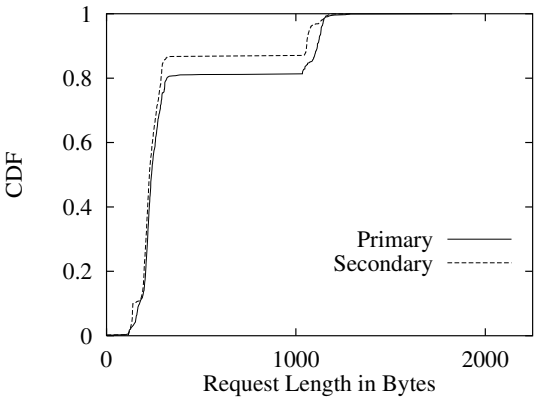
Figure 3.8b shows the result of dividing one of the request length distributions by by primary and secondary requests. The result yields two significantly different distributions, and therefore Mah chooses to include this division of requests in the HTTP-traffic model.

Reply Length

The *reply length* distribution describes the length of the Web objects, such as HTML documents or images, being transferred from the Web server back to the client. Figure 3.9a shows the CDF for the reply length distribution for each of the traces.



(a) CDFs of HTTP Request Lengths [40]



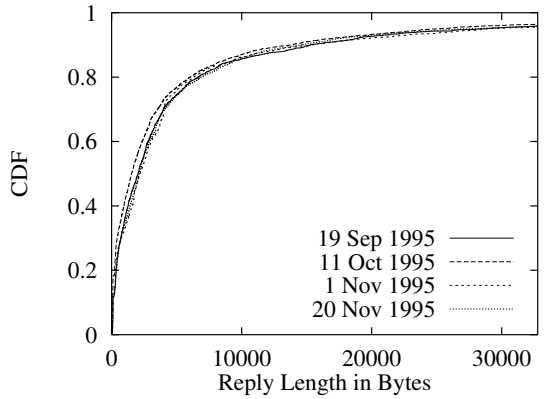
(b) CDF of primary and secondary request lengths, 19 September 1995 [40]

Figure 3.8: *Request length distributions.*

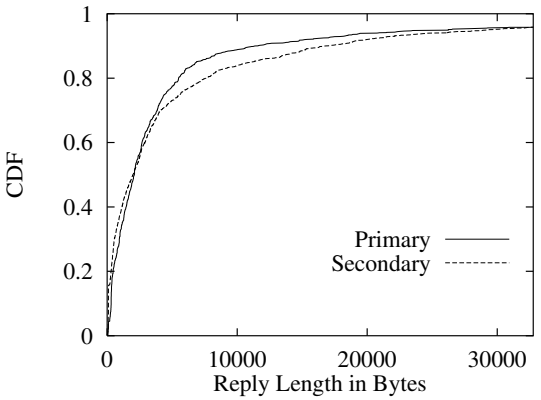
From the plot we can see that the median reply length lies around 1.5-2 KB and that approximatly 95% of all the reply lengths are smaller than 30KB. Furthermore it was found that in each of the traces, reply lengths larger than 1MB were present.

By performing an analysis of reply sizes equivalent with the one performed by Crovella and Bestavros in [12], Mah found the behavior of the distribution of requests larger than 1KB to be "reasonably well modeled" with a Parato distribution [40]. The α value estimate in Mah's estimation ranges from 1.04 to 1.14, while the estimated α value in [12] was 1.06.

As with the request length distribution, Mah found it appropriate to divide the reply size distribution into a primary and a secondary distribution. Figure 3.9b shows the CDFs for reply length distributions for the trace taken the 19. September 1995.



(a) CDFs of HTTP Reply Lengths [40]



(b) CDF of primary and secondary reply lengths, 19 September 1995 [40]

Figure 3.9: *Reply Length Distributions.*

Server Selection

The *server selection* distribution describes the relative frequency with which each Web server is visited for some set of consecutive document retrievals. Mah was unable to derive this distribution from the traces since the majority of traffic in the traces was to the local department Web server. Instead Mah used a Zipf's Law distribution [72] to describe the popularity of different Web servers. This is a heavy-tailed distribution and thus reflecting the fact that some sites are many times more popular than other sites.

Summary

The Mah model model gives a very detailed description of the main quantities in Web traffic. It is based on a quite small data sample. However, we feel that the key elements for a Web traffic model are present. Especially the evidence that the reply length distribution is heavy tailed strengthens credibility of the model because

this is the primary requirement for the ON/OFF model. Furthermore, because the model gives such a detailed description of user behavior, we find very it suitable for modeling the individual browsing user, thus allowing us to measure the performance experienced by the modeled users.

A general problem with traffic models such as the one developed by Mah is that they are only credible for a certain amount of time because traffic characteristics may change as new applications become popular on old applications may be used in different ways. Again we claim that the presence of heavy-tailed characteristics provides a strong indication that the model has the key characteristics of more recent models. However, the Web is very dynamic and the different distributions of the model have certainly changed since 1995, and thus this should be an aspect to remember when drawing conclusions on experiments with this traffic model.

3.4 Implementation

Based the techniques described in [4] and [14] we have successfully implemented a traffic generation tool that efficiently models a user browsing the Web, allowing us to generate a high aggregate traffic load produced by modeling hundreds of simultaneously browsing users. In practice the tool consists of a *browser program* that simulates browsing users, and a *server program* that responds to requests generated by the browser program.

With the aspects of traffic generation and traffic behavior in mind, we set the following high level goals for the traffic generator:

- The traffic generator should produce a realistic traffic workload, that is, the traffic should be generated based on an well understood empirical or analytical model of Web traffic as described in the previous sections.
- Given that sufficient resources are available, the traffic generator should always exhibit the same behavior independent of any surplus in resources, both counting CPU power or network bandwidth, that may be available.

3.4.1 Performance Measurements

A central aspect of our study of RED queue management performance is that we want a user centric performance measure. In the case of Web traffic we measure the response time. That is, the time it takes to retrieve Web objects from servers. This measure reflects how changes in a network configuration impacts the performance experienced by a user using the network.

The traffic generator is instrumented to measure the response time experienced by each modeled user. The response time is defined as the elapsed time between the time of the socket `connect()` operation and until the client has received the data requested from the server and reads an EOF marker. Figure 3.10 describes the response time using a message sequence chart of a request for a Web object.

Following the messages sequence chart we can see that the request begins when the client opens a connection to the server by sending a SYN packet. After TCP's initial three way handshake the connection is established and the client sends the request data. The server responds to the request when all the request data has been received from the client, by sending the reply data. Once the server receives an ACK on the last data sent to the client it performs an active close on the connection. The response time is the time from the initial SYN is sent from the client until the client receives the FIN from the server. Note that there are no drops or retransmissions in the illustration, and acknowledgements are only shown for connection establishment and termination.

It should be clear that we do not include the entire four way handshake of the connection termination in the response time measurement. The tool models HTTP 1.0 traffic. Therefore each element of a Web page is retrieved in a separate TCP connection. This means that response time is the time it takes to retrieve an element of a Web page, not the time it takes to retrieve the entire page.

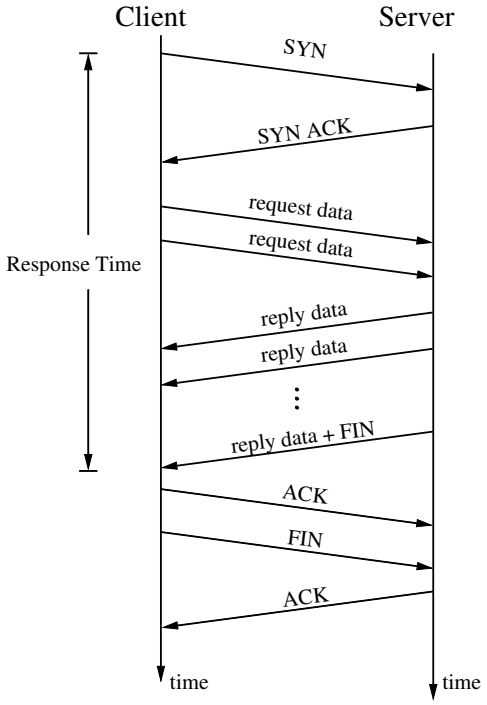


Figure 3.10: *Message sequence chart of a Web object request.*

Page response times are also measured by the tool. Page response time is the time from the first request is issued until the last reply has been received. This makes it possible to compare the performance between different techniques for transferring Web pages. For instance one may be interested in comparing the performance using persistent connections versus non persistent connections. In addition to the response time measurements, the tool associates relevant information such a the reply size, and connection IP and port numbers to each measurement, allowing association of response times to for instance reply sizes or specific hosts in the network.

3.4.2 Protocol

To simplify the implementation as much as possible to ensure efficiency, the traffic generator only "simulates" HTTP though an extremely simple protocol. This means that TCP connections are established and data transferred, in amount prescribed by the Mah model, but without the actual HTTP requests or replies. Instead the client simply sends a request containing the amount of data it expects returned from the server. The server responds to client requests by sending the specified amount of data. A benefit of this design is that we avoid having to use an actual Web server to respond to the requests, thus avoiding the processing overhead of having a much more general server. Tools like SURGE [5] and SCLIENTS [4] implements a traffic generator that make actual HTTP requests that can be parsed by a Web server, thus making the tools applicable for performance testing of Web servers. However

our focus is to generate Web like traffic on a backbone network, not to build tool for performance testing Web servers.

3.4.3 Multiplexed I/O and TCP Configuration

The client side of the traffic generation program needs to model hundreds of actively browsing users simultaneously. Each model can have several open TCP connections to a server, and at any time transfer data or pause while waiting for the think time to timeout. To implement this efficiently we relied on experiences described in [4], where the authors describe how I/O multiplexing was used for building the SCLIENTS tool. We have adapted the same technique for our traffic generation program, and again, it has proved to be an efficient method for implementing a traffic generator. One drawback of this method is that one must be careful that the application does not block for longer periods for time, leaving other connections unserviced for too long. Another drawback is that the traffic generation tool operates in a polling loop so the tool always monopolizes the CPU of the local machine.

To ensure optimal performance we have disabled the Nagle algorithm in the traffic generator. The Nagle algorithm has been included in TCP to optimize bandwidth utilization on the Internet [49]. The idea is to try to minimize the overhead of packet headers by increasing the average size of the payload in TCP packets. Thus, if a TCP sender only has a small amount of data (less then Maximum Transfer Unit) ready to be sent, then it should wait until all previously sent data has been acknowledged by the receiver, before sending the packet. As described in [47] using Nagle in an application can significantly decrease the network performance.

3.4.4 Modifications and Bugs

In the actual implementation we decided to limit the duration of the think times to 10 minutes. This corresponds to the 90% quantile of the think time probability distribution. The reason that we want to avoid having browser instances remaining passive during the entire time the traffic generator runs. Another way to do this could be by scaling the think times, as was done in [53]. A second modification is that we do not use the Zipf's Law distribution when choosing Web servers. Instead the servers are chosen uniformly, so that each server will receive approximately an equal number of requests.

The implementation has a number of unintentional differences from what we have just described. The first problem is that, by accident, the first two reply sizes are always samples from the primary reply size distribution. This makes the reply size distribution slightly different from the distribution reported in the Mah paper. However, we believe that this does not significantly impact generated traffic. The second problem is that we do not have any think time when the modeled browsing user moves from one server to another server. This has the effect that clients will generate more traffic, than the clients would do if they followed the definition of think time described earlier. Again we do not believe that this significantly alters the traffic generated by the tool, except that we would need to model more browsing users to generate the equivalent amount of traffic.

3.5 Test of Traffic Generator

We have conducted a number of tests of the implemented traffic generator. First, experiments are performed for studying the traffic generated by the tool to confirm that it behaves as we expect, and to choose the optimal duration of an experiment. Second, we tested the tools hardware requirements by both testing whether the tool

is constrained by lack of resources on our test machines, and whether it is influenced by the amount of available resources. Finally, we performed a test that calibrates the tool so that we have a measure of offered load as a function of the number of simulated browsing users.

3.5.1 Network Setup

The experiments are conducted on the laboratory network that we describe in detail in Chapter 4 on page 49, along with a description for the experimental procedure. However, we give a brief introduction here.

The laboratory network is shown in Figure 3.11. We use 7 client machines and 7 server machines. Each machine is connected to a full-duplex 10Mbps Ethernet segment which again is connected to a switch. The switch multiplexes the aggregate traffic from all the client machines and all the server machines on to a unidirectional 100Mbps link to the opposite switch. This ensures that packet congestion and collisions cannot occur in this network. The network is instrumented so that we can measure the aggregate traffic load on the path from the servers to the clients. We refer to this network configuration as the *unconstrained network*, because the network is designed such that no traffic congestion can occur.

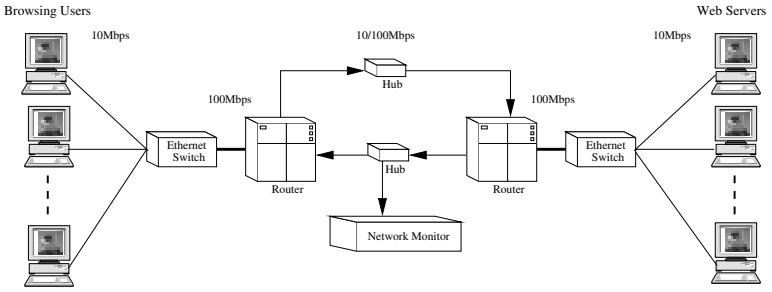


Figure 3.11: *Diagram of the laboratory network.*

3.5.2 Bursty Traffic

As we have described, the generated Web-traffic should behave as a fractional Brownian motion, e.g. it should be probabilistically self-similar and be bursty over several time scales. We check the burstiness of the generated traffic through a simple visual test.

The experiment simulates approximately 3500 browsers corresponding to an average aggregate load around 10Mbps. During the experiment each of the clients machines runs an instance of the program simulating browsing users and each of the servers runs an instance of the program simulation a Web server. The browser programs are configured to make requests to any of the servers, and the duration of the entire experiment is 55 minutes; measurements from the first 20 minutes are discarded because of startup and stabilization effects.

To illustrate the burstiness of the generated traffic we first show two figures illustrating the actual behavior of the browser programs. Figure 3.12 shows a plot of the aggregate number of requests issued by 3500 users. From the plot we see that the number of requests per second constantly varies from around 50 requests per second up to around 200 requests per second. Figure 3.13 shows a plot of the aggregate number of bytes requested per second by 3500 users. The number of bytes requested per second is constantly oscillating, varying from 250 KB per second to

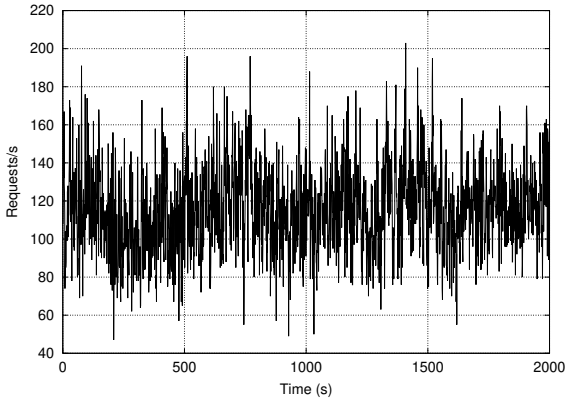


Figure 3.12: *Requests per second.*

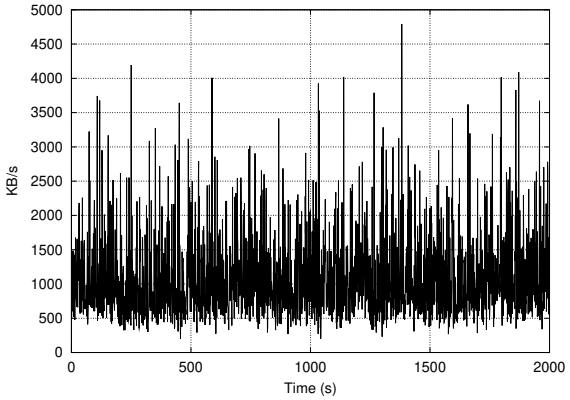
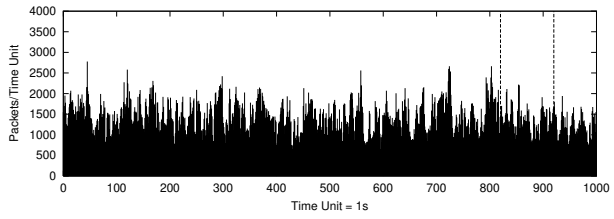


Figure 3.13: *Requested KB per second.*

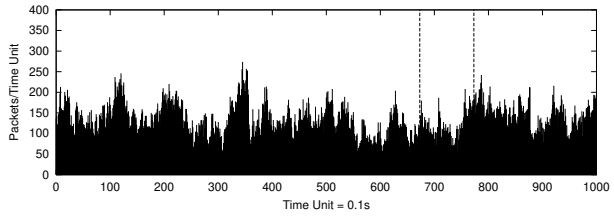
more than 4500 KB per second, giving a clear impression of how the offered load fluctuates over time.

The central indicator of burstiness in the generated traffic is the actual aggregate load produced on the backbone of the laboratory network on the path from the servers to the clients. It is shown in Figure 3.14. The plot shows the number of packets on the path from the servers to the clients over three different timescales. Figure 3.14a shows number of packets per second. Figure 3.14b is a zoom in of the marked range in Figure 3.14a from $820 - 920$s, showing the number of packets per 0.1 second. Again, Figure 3.14c is a zoom of the marked area in Figure 3.14b from $680 - 780$s averaged in periods 0.01 seconds. The stating point for each interval in which we zoom has been randomly chosen.

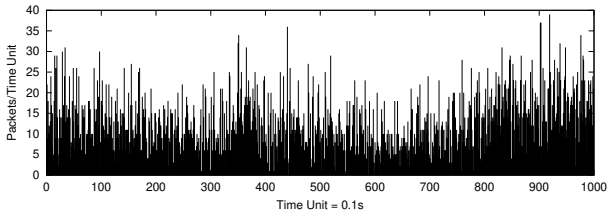
Comparing the plots with time intervals 1s and 0.1s we find that these seem to be qualitatively the same as described in [6]. The plot at 0.01s is also bursty, however the in a somewhat different way compared to the previous plots. Overall, based on Figure 3.14, we conclude that the generated traffic is bursty independent of time scale.



(a) Time Unit = 1s



(b) Time Unit = 0.1s



(c) Time Unit = 0.01s

Figure 3.14: *Packets per time unit at 3 different timescales.*

3.5.3 Bottleneck and Independence Tests

In our requirements to the traffic generator we stated that given sufficient resources, the traffic generator should behave independently of any surplus of resources. To test whether this requirement is fulfilled requires testing several different aspects: First, we need to ensure that sufficient resources are available for running the traffic generator within the range of parameter settings that we will be using. Second, we need to ensure that the behavior is identical and independent of the machine on which the traffic generator runs.

To perform these tests we have designed a set of two almost identical tests that each test the client side, and the server side traffic generator.

Browser Program

Like the previous experiment, we use the unconstrained laboratory network for these experiments. However, in this test we only use one client machine and 7 server machines. Consequently, only the machine running the browser program is a potential bottleneck in these experiments. Again each experiments runs for 55 minutes, leaving 35 minutes of useable measurements.

Since we both need to test for potential bottlenecks on the client machine and for independence of any additional resources, we test the browser on two different machines: the one with the slowest CPU, a 66MHz 80486, and the fastest, a 200MHz Pentium Pro.

On each of the two machines we run a number of experiments where we gradually increase the number of browsers simulated by the browser program, ranging from 300 browsers up to 1500 browsers.

We expect the results to show a linear relationship between the number of simulated browsers and the bandwidth utilization on the path from the servers to the client. The observation leading to this conclusion is that each browser is based on the same empirical model of HTTP traffic, meaning that each modeled browsers will on average use the same amount of bandwidth, if measured for long enough a period (35 minutes). Therefore, a linear relationship between the number of modeled browsers and the average amount of bandwidth used by the browser program should exist. If not then a bottleneck may be present or the implementation may be sensitive to the amount of available resources.

Table 3.3 lists the experiments run for the browser program test. Additionally, we have added the measured bandwidth utilization on the path from the servers to the client. To show that a linear relationship exists between the number of simulated browsers, and the bandwidth utilization, we have plotted the bandwidth measurements as a function of the number of simulated browsers. From Table 3.3

	486-66MHz	Pentium Pro 200MHz
browsers	link util (Kbps)	link util (Kbps)
500	1368	1464
650	1888	2008
800	2312	2400
950	2624	2752
1100	3008	3088
1250	3696	3552
1400	4048	4096

Table 3.3: *Browser program tests.*

and the plot in Figure 3.15 we see that a linear relationship certainly exists between the number of modeled browsers and the bandwidth utilization. Furthermore, we see that the bandwidth utilization is nearly identical independent of machine.

As a final note we should mention that we did try to simulate more browsers then the results show here, and for the 80486 machine the limit was around 1700 simulated browsers and a slightly more for the the Pentium Pro machine. Thus, we should be careful not to increase the number of simulated browsing users to a value greater then what we have tested here.

Server Program

The test of the server program is similar to the test we performed with the browser program. The only difference is that when testing the server program we run with 7 instances of the client program and only one instance of the server program.

Like the test of the browser program, we run a series of these with a low end and a high end machine. On each of these machines a series of experiments is run where the aggregate number of browsers simulated by the clients ranges from 500 to 1500 browsers. Since there is only one server available, all the simulated browsers will only use that server.

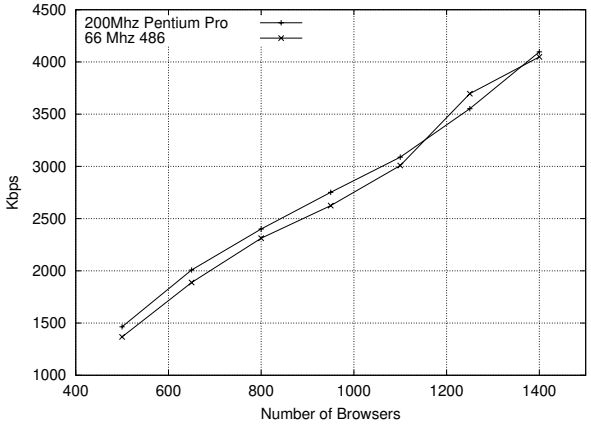


Figure 3.15: *Browser program test.*

A possible problem with the general setup where we run with several browser programs and server programs, is that there is a risk that one server at some point is significantly more popular then the other servers in the experiment. Such a situation could cause a peak load on the server to be higher than what we test for here. We see this as a minimal risk since servers are chosen uniformly, and because the load in this test is nearly twice the average load during the actual study.

The experiments and the results are summarised in Table 3.4. Figure 3.16 shows the plot of bandwidth utilization as a function of simulated browsers. From

	486-66MHz	Pentium II 400MHz
browsers	link util (Kbps)	link util (Kbps)
100	307	314
300	840	888
600	1680	1712
900	2664	2512
1200	3456	3440
1400	4040	4040

Table 3.4: *Server program test.*

the results we see that the server program is not a bottleneck for the performance of the browser program within the range of browsers we simulate, and furthermore, that the performance seems to be independent any surplus of resources available on the machine running the server program.

3.5.4 Calibration

In this final set of experiments we calibrate the browser program, to obtain a function describing the offered load from the traffic generator as a function of the number of simulated browsers. The *offered load* is the load the traffic generator will generate without constraints from CPU or network resources.

For these experiments we also use the laboratory network configured without bandwidth constraints. We use all seven client machines and all seven server machines. Each experiment runs for 55 minutes leaving 35 minutes of simulation

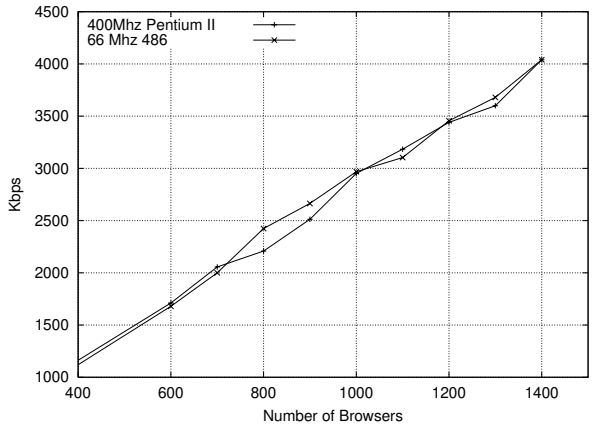


Figure 3.16: *Server program test.*

measurements.

In each experiment we increase the number of modeled browsers ranging from 700 to 5200. In each experiment we measure the bandwidth utilization on the path from the servers to the clients. Thus, the bandwidth measurements only reflects the load produced from Web replies, and not requests.

Figure 3.17 shows the results of the measurements and a linear approximation of average bandwidth utilization as a function of the number of simulated browsers.

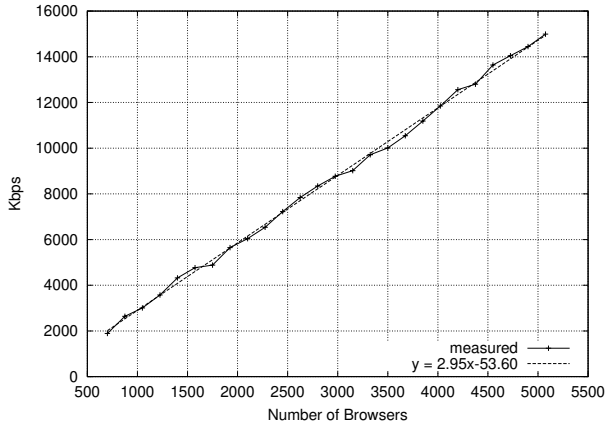


Figure 3.17: *Offered load as a function of the number of simulated browsers.*

3.5.5 Summary

We have conducted a number of experiments in which we ensure that the implemented traffic generator performs and behaves as predicted. We have demonstrated how the generated traffic is bursty over several time scales. This test was followed by tests that showed that the traffic generator is independent of CPU resources within the range of browsers that we intend to simulate in our experiments.

In a final set of experiments we have calibrated the traffic generation tool, thus

Link utilization (Mbps)	Number of browsers
5.0	1714
7.0	2389
8.0	2727
9.0	3065
9.8	3353
11	3740

Table 3.5: *Typical offered load levels used and number of simulated browsers.*

providing us with a load measure describing the offered load as a function of the number of simulated browsers.

3.6 Conclusion

In this chapter we started by describing previous work on traffic generators. Then we looked closer at studies describing the statistical properties of Web traffic and how Web traffic can be modeled parsimoniously using the ON/OFF model with a heavy-tailed ON or OFF distribution.

We then described the HTTP traffic model developed by Bruce Mah, which we have chosen to used in our traffic generator. Following this we described how the traffic generator has been implemented and how we measure HTTP response times.

Finally we have shown the results of testing the traffic generator tool. By first demonstrating how the traffic is bursty, and then showing that there are no bottlenecks when using the traffic generator on our unconstrained network. Finally we showed the results of a series of calibration experiments which gives us a measure of offered load as a function of simulated browsing users.

Chapter 4

Network Design and Experimental Procedures

In this chapter we describe the laboratory network on which we perform the experiments for studying the impact of using RED. In addition, we describe the experimental procedures, and furthermore, the results of two experiments are presented. We use these to illustrate how we compare performance experimentally.

4.1 Network Design

The central issues in the design of the laboratory network is to provide a environment in which we can measure the impact of changing the queuing mechanism or its parameters. It is important to realize that the laboratory network must provide an environment such that the traffic generator generates a realistic workload. Only high level protocol events are modeled in the traffic generator, the remaining events are created dynamically and in some sense controlled by the network environment in which the experiment runs.

4.1.1 Network Model

We aim at modeling a campus or small enterprise network with a single wide-area link to an Internet Service Provider (ISP) as illustrated on Figure 4.1. Starting on the left side of the illustration, we have a local area network (LAN) for the campus or small enterprise. Traffic, generated by browsing users, from the campus or enterprise network is carried to the Internet (on the right) through two routers. One router is located at the edge of the local network and the other at the edge of the ISP network.

The laboratory network depicted in Figure 4.2 models the scenario that we have described. Notice that each Ethernet segments has been labeled E_1 to E_6 and routers have been labeled R_1 and R_2. On the left side we model the LAN of the campus or enterprise network (E_1) by simulating a number of client machines that run instances of a browser program that issues Web requests. At the other side we model the Internet (E_2) by simulating a set of server machines running instances of the Web response generator. The LANs on each side of the network are switched, and each host is connected to the network switch through a separate 10Mbps Full-Duplex link.[1].

[1] To be precise, we use only one physical switch, but by using Cisco VLAN technology we divide the switch into several separate LANs.

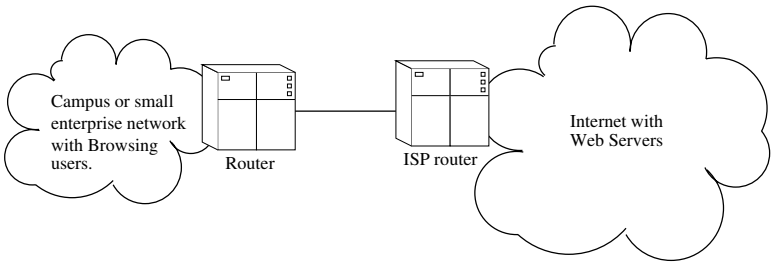


Figure 4.1: *The scenario modeled by the laboratory network.*

At the core of the laboratory network are the two routers R_1 and R_2 connecting the LANs at the edges of the network. Each router has one 100Mbps Full-Duplex Ethernet interface attached to the LAN from which it is forwarding data (E_3 and E_4). Each router also has two additional 10/100Mbps Ethernet interfaces configured to create two point-to-point Ethernet segments E_5 and E_6 (using two hubs) that connect the routers R_1 and R_2. Static routes are configured on the routers such that all traffic flowing from the servers to the browsers use Ethernet segment E_5 and all traffic in the opposite direction use the Ethernet segment E_6.

Depending on the network configuration of the two routers, the laboratory network is configured to operate either in *constrained* mode or in *unconstrained* mode. To operate in constrained mode, we configure the router-to-router Ethernet segments to run at only 10Mbps. Our representation of the ISP link is a potential bottleneck since the aggregate bandwidth available to the machines at each edge of the network is limited only by the 100Mbps links from the LANs to the routers. When the links connecting the routers are configured to run at 100Mbps, this potential bottleneck is removed and we run in unconstrained mode.

The router machines machines run Alternate Queueing (ALTQ) version 1.2 [36] extensions to FreeBSD. ALTQ extends the network-interface output queuing discipline to include, among others: tail-drop, RED, CBQ, and WFQ queue-management disciplines. Appendix C describes a small difference between the Floyd and Jacobson [29] definition of RED and the ALTQ implementation, and how we repair it.

The load on the core of the network is highly asymmetric. This is because Web responses are generally much larger than Web requests, and therefore traffic from servers to clients consumes several times more bandwidth than the traffic in the opposite direction. On the constrained network this means that only the outbound interface on the ISP router carrying traffic to the browsing users can be congested. Consequently, it is only on this interface that the effects of different queue management algorithms on the IP output queue will show. Therefore, it is only on this interface that we operate with different queue management mechanisms and configurations, see Figure 4.2. IP output queues for the link interfaces on all other machines in the network are tail-drop queues with the FreeBSD default queue size of 50 elements.

Appendix A contains a detailed network diagram and describes the hardware we use in the network.

4.1.2 Modeling Latency

Another important factor in modeling this configuration is the effect of the end-to-end latency. Apart from being a fundamental element of the Web, latency has a significant on the behavior of TCP. For instance the latency controls the speed

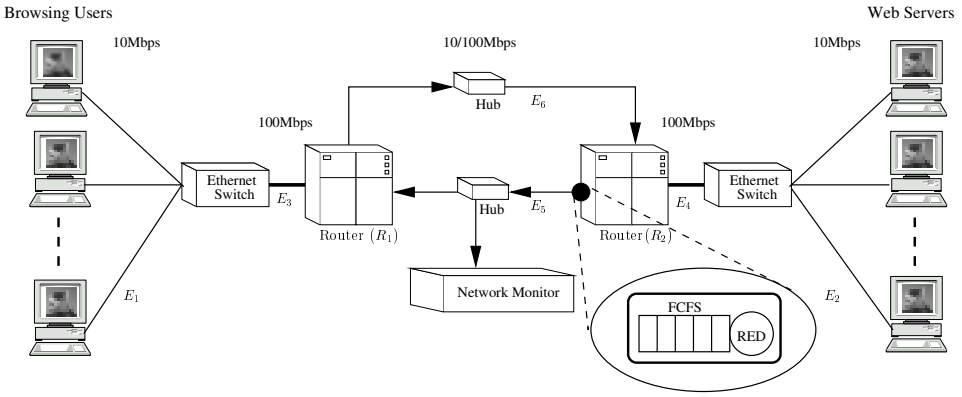


Figure 4.2: *Laboratory network. Each Ethernet segment is labeled E_1 to E_6, and routers are labeled with R_1 and R_2.*

	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

Table 4.1: *Round-trip times in milliseconds between pairs of machines.*

at which the transmission window can grow because this is controlled by the rate at which packets are acknowledged. Furthermore by modeling latency we avoid potential synchronization effects in the laboratory network.

We use the *dummynet* [60] component of FreeBSD to configure in-bound packet delays on the hosts running the Web server program. This allows us to emulate different round-trip times between each pair of a browser machine and a server machine. Table 4.1 gives the delay between each pair of machines. Figure 4.3 shows a histogram of round trip times. The values we use are taken from measurements obtained at the *NetStat.net* web site [51] and were chosen to represent a sample of Internet round-trip times within the continental U.S. Using these values, the minimum round-trip time experienced by an arbitrary TCP connection will be one of these values depending on which pair of machines makes the connection, assuming no delays in the two routers. As described in Section 3.4 on page 38, the browser program chooses a server based on a uniform distribution, and thus, each of the round-trip time values is represented equally and we can therefore calculate the mean nominal round-trip time for all TCP connections sharing the network as approximately 79ms.

4.1.3 Monitoring

As we described in Section 3.4.1 on page 38 the traffic generation tool is instrumented to report response time measurements. To gain further insight in the general performance of the laboratory network, we have instrumented the laboratory net-

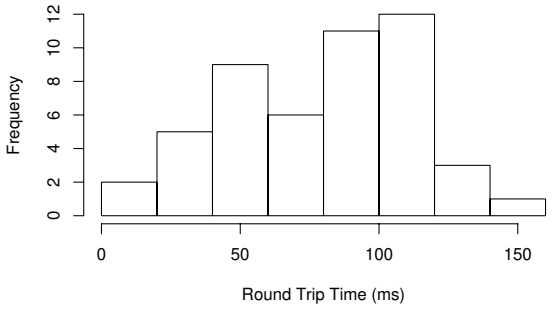


Figure 4.3: *Histogram of round-trip times between pairs of machines.*

work to collect general statistics of bandwidth utilization and packet drops during experiments.

The monitoring is done by two programs. The first program monitors the behavior of the queue management algorithm. When running it reports a summary of events every 100ms. This summary consists of the number of dropped/forwarded packets, minimum and maximum queue length since the last summary. Additionally, the program reports the mean and variance of the queue size based on samples of the queue length taken every 3ms.

The second monitoring program runs on an external machine connected to the hubs forming the link between the routers. It collects the TCP/IP headers in each frame traversing the link and processes these to produce a log of link throughput over each specified time interval (typically 1s).

To ensure that no unexpected buffer overflows in any other parts of the network, we also monitor the IP packet queue of each network interface in the network, using the *netstat* command. Furthermore we collect traffic summaries from the switch which among other facts shows whether it has dropped any packets.

4.1.4 Operating System and Protocol Configuration

All machines in the network are FreeBSD 2.2.8 machines with TCP Reno implementation that support the timestamp feature described in RFC 1323 [33]. Window scaling is disabled and SACK is not included in the implementation. The default window size of 16KB is used. To increase the timer resolution, all kernels are configured to run at 1000Hz.

As described in Section 3.4.3 on page 40, the traffic programs have the Nagle algorithm disabled to avoid the additional time added to transactions.

4.2 Experimental Procedures

Each experiment uses the following procedure: After initializing and configuring all router and end-system parameters, the server-side processes are started followed by the browser processes. Each browser process emulates an equal number of users chosen, to place a nominal offered load on an unconstrained (100Mbps) network. The offered loads we have chosen to use in the experiments are: 50,70,80,90,98,110 percent of the capacity of the 10Mbps links 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 load. Because the measured response times at a load of 110% had deteriorated well beyond levels that most users would tolerate, we have decided to not consider loads beyond 110% on the congested link.

Each experiment runs for 90 minutes, but data collected during the first 20 minutes are discarded in the reported results to eliminate startup and stabilization effects. These effects are shown in Figure 4.4 which is a plot of mean response times of all requests averaged in one second periods in a typical experiment.

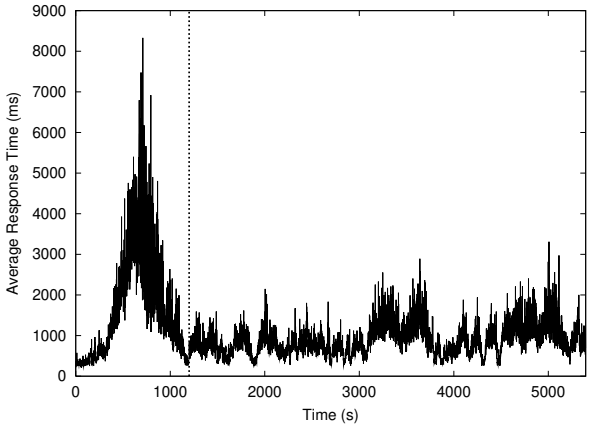


Figure 4.4: *Average response time per second during an experiment. The plot includes the initial 20 minutes, where traffic generators are in the initial startup phase (marked by the vertical dotted line).*

4.3 Performance Evaluation

Each completed experiment is described though a set of summary statistics describing the performance. As we stated in the introduction, our focus is to determine the impact that RED has on the end users experience of performance. Thus the primary indicator of performance is the response times measured by the traffic generator during each experiment. The response times provide a complete view of the performance experienced by the browsing user. However, they provide little information about the actual cause of good or bad performance. Therefore we supplement the measurement of response times with more traditional measures of performance such as link utilization and packet drop rates.

In the following we present the summary of two experiments. The first experiment runs at 98% offered load on the 10Mbps network (3353 browsers) on the unconstrained network with tail-drop queue management. The second experiment runs with the same offered load, but now we use the constrained network and RED queue management. We use these to show how we compare performance between experiments.

4.3.1 Experiment: unconstrained (100Mbps) network

This experiment was run on the laboratory network configured to provide unconstrained bandwidth. This is equivalent to the setup used for the calibration exper-

iments that we presented in the evaluation of the traffic generator in Chapter 3 on page 25. The experiment runs with 3353 browsers, generating a 9.8Mbps average load for the entire experiment (after the startup period).

As we described in section 4.1.1, using the unconstrained network no packets are dropped for any connection between browsing users and Web servers, and no significant queues builds up during the experiment, because there is a significant excess of bandwidth available in the network. Consequently the HTTP response times measured in this experiment corresponds to the best-case performance, independent of queuing mechanism, that one can achieve on our laboratory network.

Overall this measurement of best-case performance will serve as a reference point when evaluating the performance of experiments run on the constrained network, where congestion can occur.

The response times measured by the traffic generators is the central result of running an experiment. To present these without too much loss of information we plot Cumulative Distribution Functions (CDFs). This is shown in Figure 4.5 which shows the response time CDF for this experiment. From this figure we conclude that about 90% of the requests complete within 500ms or less.

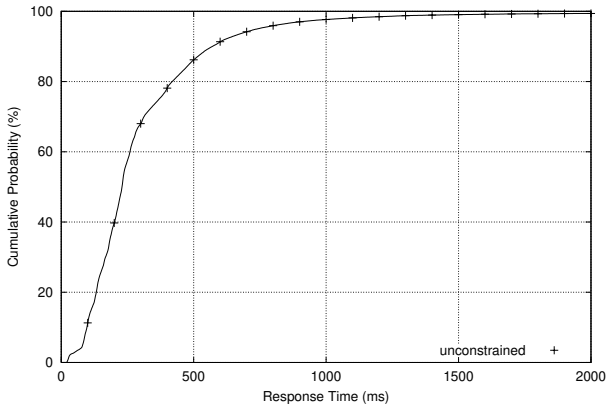


Figure 4.5: *Response time CDF for the experiment on the unconstrained network.*

The additional performance measurements are summarized in Table 4.2. The first field shows the average number of KB sent per second on the link from R_2 to R_1 during the experiment. The next field shows the percentage of packet drops for all packets arriving at router R_2 from LAN L_2 on which the server programs are running. The mean (μ) queue length is the mean of the mean queue lengths that are reported every 100ms (see Section 4.1). The median (η) response time is the 50^{th} percentile of the measured response times. Finally, we group the response times into three intervals, and report the percentage of requests in each interval: $0 \leq i_1 \leq 1$, $1 < i_2 \leq 2$, $2 < i_3 \leq 3$, and $i_4 > 3$ seconds.

The most interesting information in this particular table is that there are no packet drops during the experiments, and the mean queue length on the interface is zero. This is a clear indication that the link from the servers to the clients was indeed unconstrained during the experiment.

4.3.2 Experiment: constrained network with RED

The second experiment was performed on the constrained network. As with the first experiment, the load was 98% (3353 browsers). The queuing mechanism used

$\frac{KB}{s}$	% drops	μ qlen	η response time	i_1	i_2	i_3	i_4
1187	0.0	0.0	229	97.7	1.74	0.19	0.40

Table 4.2: *Performance summary of an experiment with 3353 browsers on the unconstrained network.*

for this experiment is RED using the default parameter settings from ALTQ, see Table 4.3.

name	value
qlen	60
min_{th}	5
max_{th}	15
max_p	1/20
w_q	1/512

Table 4.3: *Default RED parameter settings in ALTQ.*

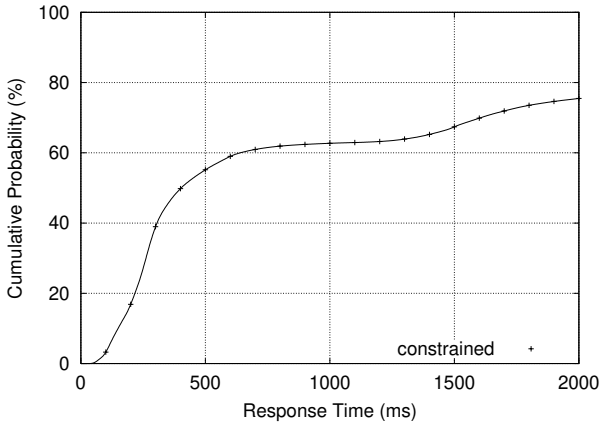


Figure 4.6: *CDF for response times for an experiment on a constrained network.*

As before, the primary result from the experiment is the response time CDF shown in Figure 4.6. Table 4.4 provides summary statistics for the experiment. Two additional columns that only apply to RED experiments have been added compared to Table 4.2. They divide packet drops into two categories: unforced drops and force drops, as shown in Figure 1.4 on page 5. Unforced drops are packets dropped early by the RED queuing mechanism, e.g. when the weighted average queue length is between min_{th} and max_{th}. Force drops are packets dropped when the weighted average queue length of RED exceeds the maximum threshold or when the actual queue length exceeds the queue length. Thus, looking in Table 4.4 we see that 11.2% of all arriving packets were dropped; of these, 43.3% were dropped by the early drop mechanism in RED, while 56.6% were force drops.

$\frac{KB}{s}$	% drops	% unforced	% forced	μ qlen	η resp. time	i_1	i_2	i_3	i_4
1142	11.2	43.4	56.6	12.2	402	62.7	12.8	9.56	14.9

Table 4.4: *Performance summary at 98% offered load on the constrained network using RED with default parameters.*

4.3.3 Comparing Response Time Performance

To demonstrate how we compare the performance between experiments we now compare the measured performance of the two experiments described in the previous two sections.

The response times CDFs are our primary base for comparing experiments. Figure 4.7 shows the response time CDFs for the two experiments in one plot. From this it is clear that constraining the bandwidth significantly impacts the response time performance for the generated Web traffic.

Another approach could be to assume similar distributions and identify the parameters of the distributions. This would allow us to use well established methods for comparing performance such as mean, confidence intervals and so forth. However, the distribution of response times is quite complex and cannot be well modeled by a single distribution. Consequently, an analytical approach would require a significant amount of statistical analysis without necessarily improving the quality of the study at hand.

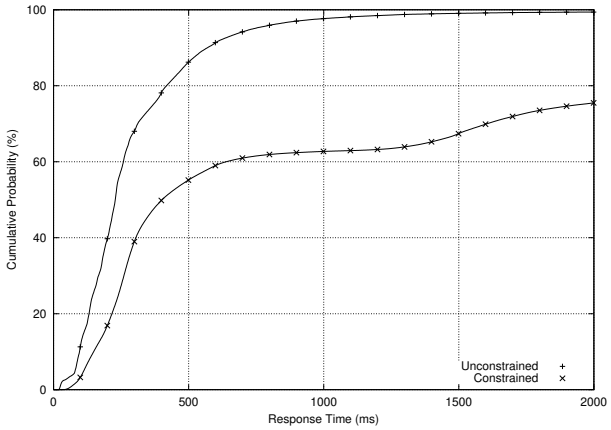


Figure 4.7: *Comparison of response time performance.*

A potential side effect of running an experiment on the unconstrained network is that since the average request takes longer to complete, the total number of requests completed on the unconstrained network may be significantly larger than the equivalent experiment on the constrained network. In fact, the same problem could occur when comparing two experiments with different parameter configurations on the constrained network. Therefore we always ensure that this is not the case when comparing experimental results by comparing the number of completed HTTP requests in each experiment.

As a reference, Table 4.5 shows the number of requests generated during a 70 minute interval for each of the loads in typical runs on the unconstrained network.

Experiment	Load %	Requests
1	50	240,379
2	70	329,638
3	80	375,673
4	90	425,293
5	98	461,837
6	110	521,561

Table 4.5: *Typical numbers of requests in a 70 minute interval for each load.*

In total we find our approach for comparing performance between experiments to be sufficient for studying the impact on HTTP response times when using RED queuing. However, we should be careful not to draw conclusions on marginal performance differences.

4.4 Summary

In this chapter we described the laboratory network and the experimental procedures. This is followed by the results of two example experiments for which we demonstrate performance measurements and our procedure for comparing response time performance between experiments.

The laboratory network models a typical scenario in which routers operate. A campus or small enterprise network is connected to an Internet though a link to a Internet Service Provider (ISP); this link may be congested when users on the campus or enterprise network request Web pages that are placed on Web servers located somewhere beyond the ISP.

To allow comparison of performance with and without bandwidth constraints, the laboratory network can be configured as a *unconstrained* network or a *constrained* network, depending on whether there is a potential bandwidth bottleneck in the network.

Programs are used for monitoring the performance of the network, thus giving detailed information on the behavior of the queuing mechanism on the potentially congested router, and providing summary statistics about the bandwidth utilization in the network.

The experimental procedures were described, and to illustrated them, we showed the results of running experiments on the unconstrained network and on the constrained network. The main performance measure of each experiment a cumulative distribution function (CDF) representing measured HTTP response times.

The response time measurements from these experiments were by performing a visual comparison of the CDFs from each experiment. We find this method for comparing performance sufficient, but acknowledge that we should be careful when comparing experiments with marginal performance differences.

Chapter 5

Tuning RED

This chapter presents the results from our study of using RED on interface queues carrying Web traffic. The strategy for the experiments is first to examine the effects of tail-drop queuing behavior in the laboratory network with the goal of finding queue lengths resulting in good HTTP response time performance. Then we examine the effect that RED parameter settings have on HTTP response time performance and again we find the parameters resulting in good performance. Having a good understanding of the effect of tail-drop and RED parameters on HTTP response times in the laboratory network, allows us to compare the effect of using either.

Based on the analysis we learn that at loads near or below the levels of link saturation (90% or less), there is little difference in the end-to-end response time between the best-tuned RED and tail-drop configured with 1-2 times the bandwidth-delay product in buffer space. At offered loads approaching the link saturation (above 90%) RED can be carefully tuned to yield performance somewhat superior to a properly configured tail-drop. However, when offered loads exceed the link capacity, we cannot detect any difference in response time performance between the two mechanisms.

5.1 Tail-Drop Results

To establish a baseline for evaluating the effects of using RED, we examine the effects of tail-drop queuing in our laboratory network. For these experiments we use the constrained network, where a bottleneck link is created between the two routers by configuring the two segments connecting the router machines to run at 10Mbps using 10Mbps hubs, as described in Section 4.1 on page 49.

5.1.1 Methodology

The critical parameter for a tail-drop queuing mechanism 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 [32]. The guideline that appears to have become most popular is to provide buffering approximately equal to 2-4 times the bandwidth-delay product of the link. Bandwidth in this context 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 laboratory network, the mean minimum round-trip time is 79 ms, see Section 4.1.2 on page 50, resulting in a bandwidth-delay product of approximately

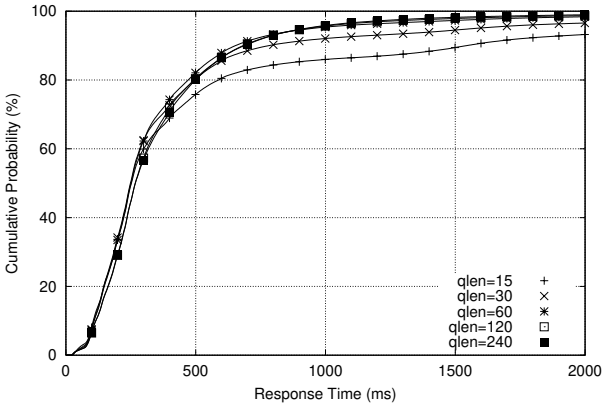


Figure 5.1: *Tail-drop performance at 80% load.*

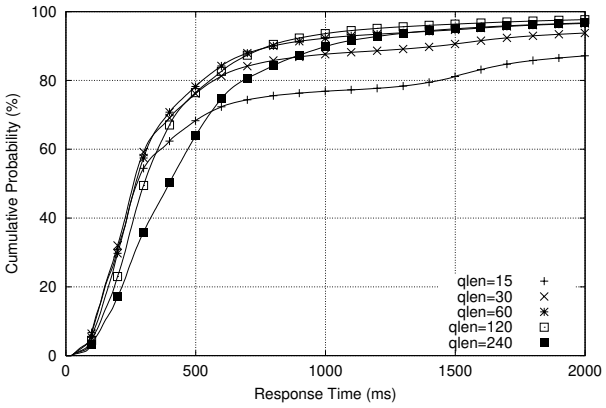


Figure 5.2: *Tail-drop performance at 90% load.*

96KB with a 10 Mbps link. 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 1KB so the tail-drop queue should have approximately 190-380 queue elements to fall within the guidelines.

The tail-drop experiments cover parameter settings with offered load values ranging from 50 up to 110% of the link capacity, and queue lengths ranging from 15 to 240 packets. A complete list of experiment configurations can be found in Appendix B.1.1 on page 93.

5.1.2 Results

We ran a number of experiments with a tail-drop queue on the bottleneck link varying the offered load and queue size. Figure 5.1-5.4 shows the cumulative response time distributions for different tail-drop 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 of 120 elements is a reasonable choice for this load.

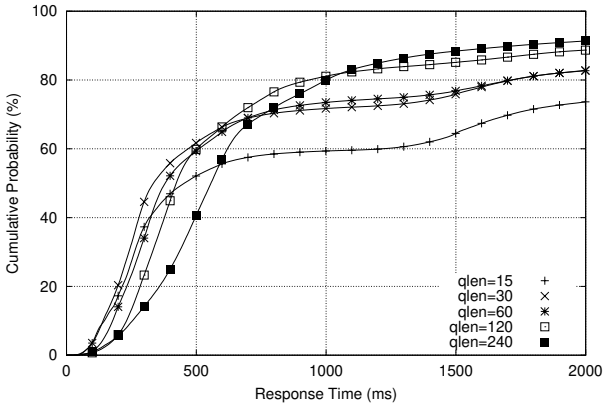


Figure 5.3: *Tail-drop performance at 98% load.*

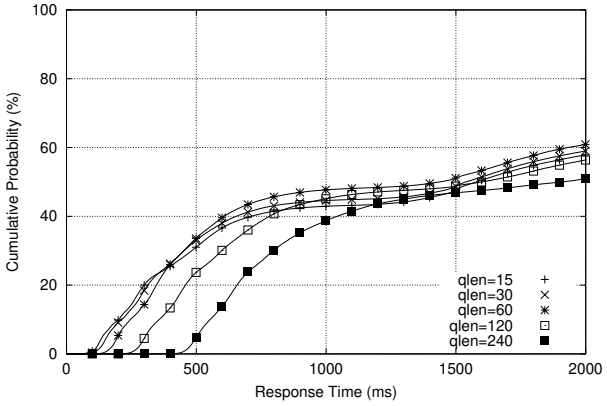


Figure 5.4: *Tail-drop performance at 110% load.*

The effect that queue size has on response times is shown on the plots for 98% load, Figure 5.3. Increasing the queue size from 30 to 120 has a slightly negative effect on requests with relatively short response times. With a queue length of 30 packets these requests complete within approximately 150-250ms, but with a queue length of a 120 packets this is closer to 250-350ms. For a 10Mbps Ethernet link and an average frame size around 1KB, approximately 1,000 packets can be forwarded per second. Thus a packet arriving at the queue already containing 100 packets has to wait approximately 100ms on the router. Such a delay is significant for requests with short responses that may otherwise complete within 200-350ms.

On the other hand, increasing the queue size from 30 to 120 significantly reduces the the number of requests with long response times that are greater than 1000ms. Even though the response time spent in the queue by each packet is longer, the reduced rate of drops means that fewer flows are 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 increased delay of short response times appears to offset any improvement gained for long response times from reduced drops.

Our results indicate that, overall, a tail-drop queue size of 120 elements (1.25 times that bandwidth-delay product) to 190 elements (2 times bandwidth-delay) is

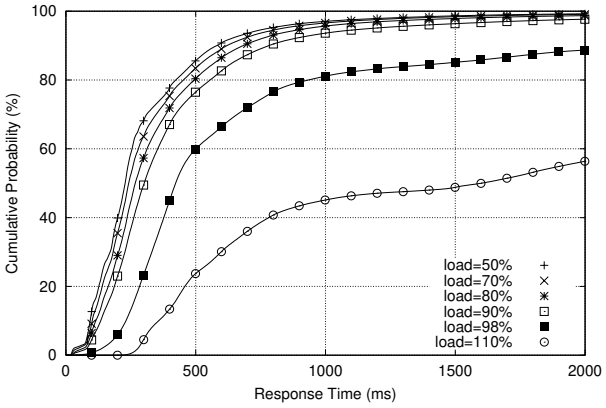


Figure 5.5: *Tail-drop performance for different loads with a queue length of 120 elements.*

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.2 on page 94. 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 5.5 shows the cumulative distribution of response times for these loads with a tail-drop 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 times 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.

5.2 RED Results

In our experiments with RED we study the effects of RED parameters on HTTP response time performance, and thereby determine the set of parameters that provides the best response time performance in our laboratory network.

5.2.1 Methodology

The RED queuing mechanism has five different parameters for adjusting the queuing algorithms 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 is 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 baseline.

From our experiments with tail-drop queuing it is clear that there is a complex tradeoff between short response times that can be completed in a few hundred milliseconds (best with a short queue) and the number of requests that complete with

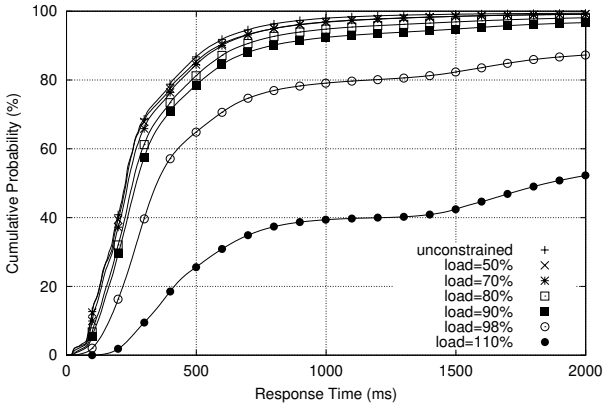


Figure 5.6: *The performance of RED at different loads.* $w_q = 1/512$, $max_p = 1/10$, $min_{th} = 30$, $max_{th} = 90$, *and* $qlen = 480$.

long response times that take more than one second (best with longer queues and lower drop rates). The original Floyd and Jacobson paper [29] suggests guidelines for tuning parameters. These have been revised based on subsequent experience and analysis (see [25] for the current guidelines). The guidelines suggest that the most fundamental effects are determined by 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 a value significantly larger than any max_{th} value used in the 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 setting was tried at five different offered loads: 50%, 70%, 80%, 90%, 98%, and 110%. See Appendix B.1.2 on page 93 for a complete list of experiments.

5.2.2 Results

Figure 5.6 illustrates typical results from these experiments by showing the effect of varying loads on response time distributions with (min_{th}, max_{th}) set $(30, 90)$. In addition we have included the response time performance measured on the unconstrained network.

At 50% load the number of dropped packets was between 0.00% and 0.01% of the total number of packets transmitted, see Table B.6 on page 103. This means that at loads of 50% and below, there is limited room for increasing the performance of the router queuing mechanism. This is also clear if we compare the performance at 50% offered load with the performance measured on the unconstrained network.

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%.

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

Exploring min_{th} and max_{th}

We start by exploring possible choices for min_{th} and max_{th}. Figure 5.7 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 [25] with a min_{th} of 5 would result in poor performance of Web-dominated traffic. The best overall response-time performance is obtained with values for (min_{th}, max_{th}) of $(30, 90)$ or $(60, 180)$. Appendix B.3.1 on page 100 shows the equivalent plots for the other threshold values.

We see, as in the case of tail-drop queuing, that there is a tradeoff between better short response times at $(30, 90)$ and reducing the number of requests with longer response times 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 the best because it improves the most noticeable delays.

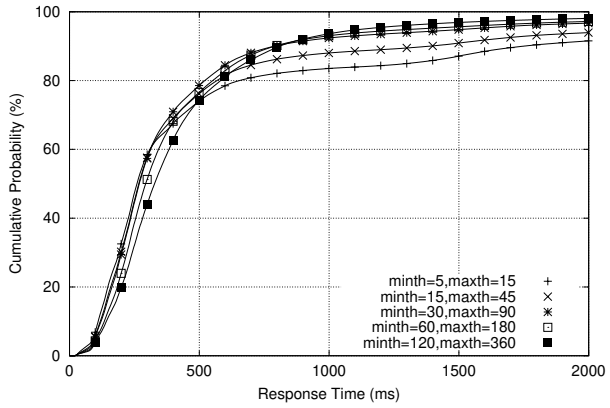
General statistics on the experiments, including link utilization and drop rates are summarized in Table B.6 on page 103. These indicate a slight drop in link utilization for the (30,90) setting over the (60,180) setting. Like tail-drop results, response times at loads of 110% are quite bad and are not 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 to 1/512, and *qlen* at 480 as in the previous experiments. Figure 5.8 illustrates the effect of 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 $min_{th} = 30$ and $max_{th} = 90$. Appendix B.3.3 on page 107 provides summary statistics and plots from additional experiments.

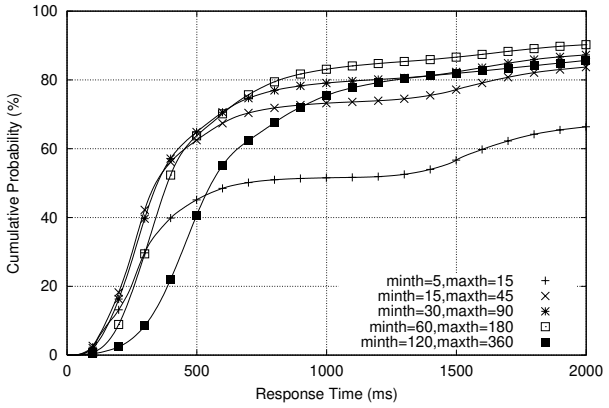
The Parameters w_q and max_p

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 wq were: 1/512, 1/256, and 1/128. (The implementation of RED requires the denominator to be a power of 2 as described in [29]). Decreasing w_q to 1/1024 was tried, but we found it 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 5.9 shows the results from the experiments at 90% load (the results at 98% load are similar). At 90% and at 98% load, the difference between the settings occurs beyond the knee (above the 75th percentile) of the CDF, meaning that changes of w_q and max_p mainly impact the number of flows with long response times. Overall,



(a) 90%



(b) 98%

Figure 5.7: *Response time CDF for offered loads of 90 and 98% of link capacity.* ($w_q = 1/512, max_p = 1/10,$ *and* $qlen = 480$).

however, we conclude that there is no strong evidence to indicate using values other than the suggested $w_q = 1/512$ and $max_p = 0.10$.

Limiting the queue size

Finally, we consider the effect of having a limit on the queue size such that there are occasionally forced drops because the queue intermittently exceeds the buffer capacity. Table 5.1 gives experimental results using our recommended values of RED parameters for actual queue sizes of 480, 160, and 120 elements.

These results are very similar to the tail-drop 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.

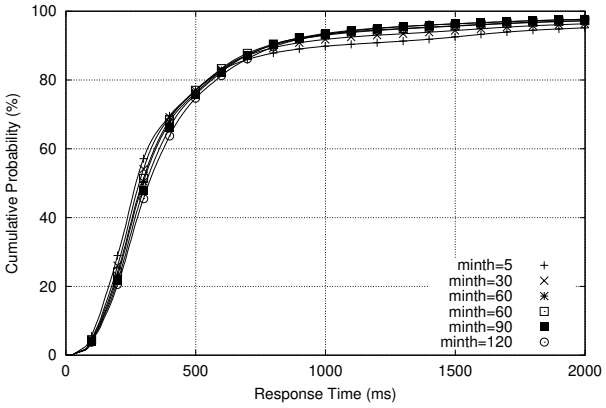


Figure 5.8: *The effect of changing* min_{th}. *Load = 90% and* $max_{th} = 90, w_q = 1/512, max_p = 1/10$, *and* $qlen = 480$.

load (%)	*qlen*	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	480	1079	0.83	20.2	269	92.4	4.30	1.98	1.32
90	160	1093	1.11	22.2	281	91.1	4.72	2.44	1.72
90	120	1066	0.72	18.8	269	92.9	4.11	1.75	1.21
98	480	1164	4.09	39.4	349	79.1	8.18	6.33	6.39
98	160	1175	5.92	46.3	401	72.3	9.70	8.20	9.78
98	120	1171	5.48	44.3	381	74.1	9.23	7.67	9.01
110	480	1187	19.7	76.0	1852	39.4	12.9	12.2	35.6
110	160	1188	19.5	76.6	1871	39.0	13.0	12.2	35.8
110	120	1188	19.4	77.4	1888	38.6	13.1	12.4	35.8

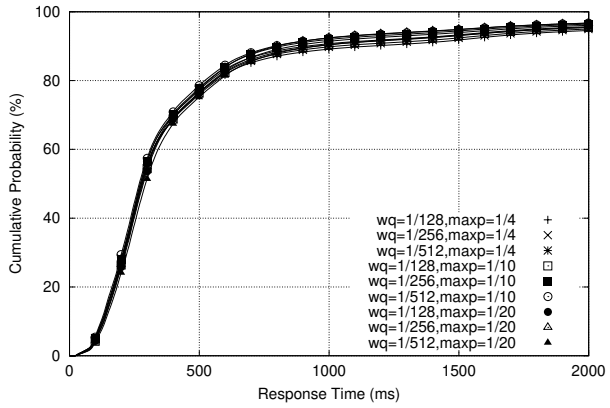
Table 5.1: *RED performance with recommended parameters and queue lengths. Remaining parameters remained fixed at:* $min_{th} = 30$, $max_{th} = 90$, $w_q = 1/512$, *and* $max_p = 1/10$.

Evaluation

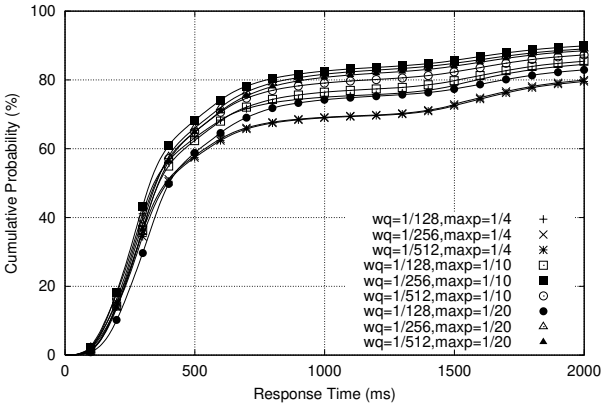
Our conclusion is that, except for min_{th} which should be set to larger values to accommodate the highly bursty character of Web traffic, the guidelines for RED parameter settings and for configuring interface buffer sizes (tail-drop and RED) also hold for the Web-like traffic used in our experiments.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

To illustrate this point, we examined the entire suite of experiments conducted for 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,
- best link utilization, and
- lowest drop rate



(a) 90% load



(b) 98% load

Figure 5.9: *Results for different values of* w_q *and* max_p. *Load = 90%, and* $qlen = 480, min_{th} = 30, and max_{th} = 90$.

where the "best" response times is a subjective choice because of the trade-off between improving for short versus long response times. The result of this search is shown in Table 5.2 and Figure 5.10 shows the response time CDFs for the found experiments. As a reference we have included the response time performance plot on the unconstrained 100Mbps network, these plots are labeled "uncongested".

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 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. There is however some degree of uncertainty in the choice of parameter settings at 98% load. The reason is that at this load we are running very close to the maximum link capacity, thus the experiments are quite sensitive to both parameter settings and the behavior of the generated traffic. As a consequence we weigh our choice of "best" parameter setting at 90% offered load highest when selecting an overall best setting.

Load (%)	min_{th}	max_{th}	w_q	max_p	Notes
90	30	90	1/512	1/10	best overall response times
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 times
98	30	180	1/512	1/10	highest link utilization
98	90	150	1/512	1/10	lowest drop rate

Table 5.2: *Empirically determined "best" RED parameter settings.*

Load (%)	min_{th}	max_{th}	w_q	max_p	*qlen*
90	5	15	1/512	1/10	480
90	5	120	1/256	1/20	480
90	120	150	1/512	1/10	480
98	5	15	1/512	1/20	60
98	5	45	1/512	1/10	480
98	5	90	1/512	1/4	480
98	120	360	1/512	1/10	480

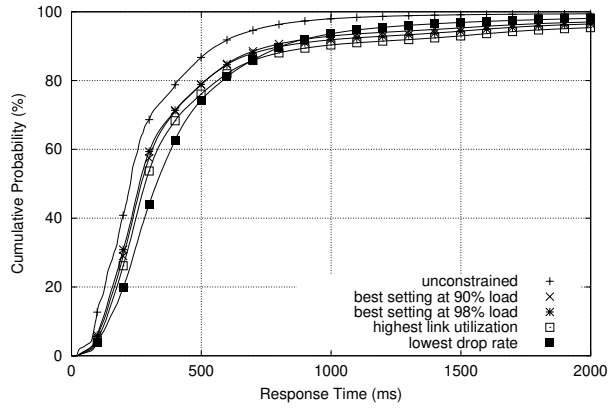
Table 5.3: *Empirically determined "bad" RED parameter values. The experiment with a queue length of 60 corresponds to the default setting of RED in the ALTQ distribution.*

There is, moreover, a significant risk of choosing "bad" parameter settings, especially at near-saturation loads above 90%. 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). The result of this search is shown in Table 5.3 and Figure 5.11. Note that plots marked with "best setting" refers to the experiment with the best overall response time that we found when searching for the "best" parameter settings.

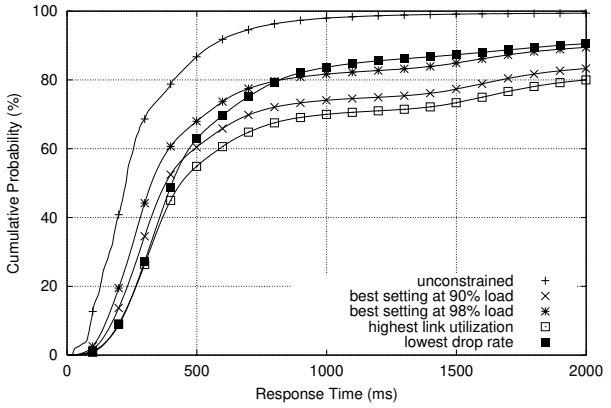
Clearly some parameter settings produce results that are considerably less desirable than our recommended ones, however, the effects of "bad" settings is most significant at 98% load. It is difficult to give a general rules for avoiding "bad" parameter settings. However, limiting the buffer size on the router, for instance by setting $(min_{th}, max_{th}) = (5, 15)$, is generally a bad idea, providing an oversized buffer is likely also deteriorate performance. However in other cases the combination of parameters deteriorate performance. This is for instance the case for the experiment at 98% load where $min_{th} = 5$ and $max_p = 1/4$. The low threshold value in combination with a rather aggressive drop probability causes many packet drops which impacts the response time performance.

5.3 Comparing Tail-drop and RED

Figure 5.12 shows the response time distributions for RED and tail-drop queuing at offered loads at 90%, 98%, and 110% with our recommended parameters selected as a result of our experiments. Also we have included the results from the unconstrained network. The only case in which there is a distinct advantage in using RED is at the 98% load where response times for shorter responses (80% of requests) are improved with carefully tuned RED parameters.



(a) 90% load



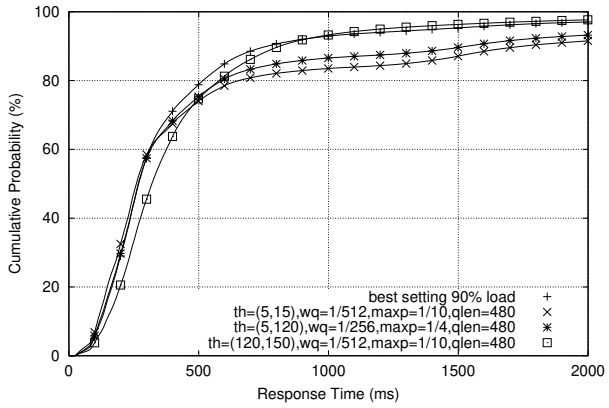
(b) 98% load

Figure 5.10: *"good" RED parameter settings.*

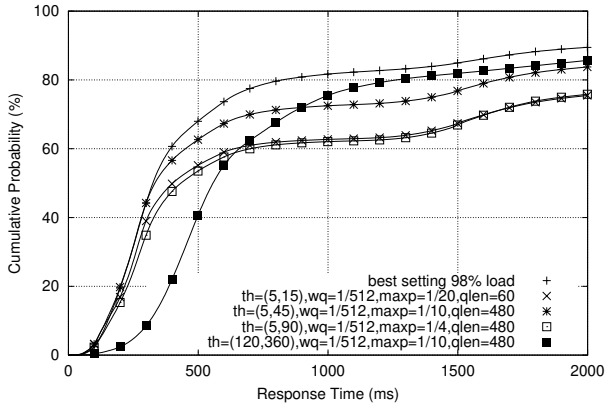
5.4 Summary

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 tail-drop queuing 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 5.11a, one can use plausible values for certain RED parameters and produce poorer performance.

At loads that approach link saturation (above 90%), RED can be carefully tuned to yield performance somewhat superior to properly configured tail-drop. In our experiments, the difference is significant only between 90% and 100% loads because response times degrade so rapidly above this level that any "improvement" from tuning RED(or tail-drop) 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 risk of choosing "bad" parameter values as illustrated



(a) 90% load



(b) 98% load

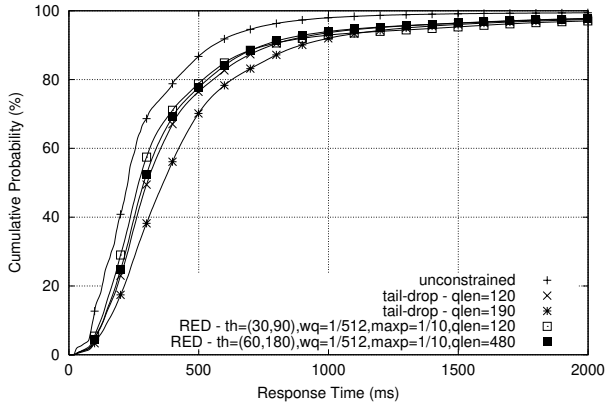
Figure 5.11: *"bad" RED parameter settings.*

in Figure 5.11b. This is important because parameter settings that outperformed tail-drop were arrived at only through extensive trial-and-error experimentation and may be quite sensitive to the scenario. It was also the case that the RED parameters that provided the best link utilization at this load produce poorer response times compared our subjective choice of "best" setting.

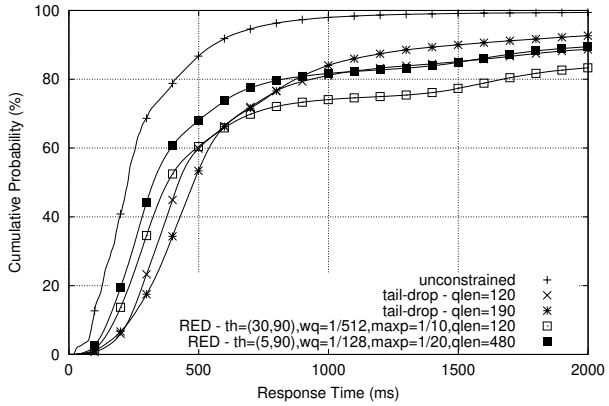
In general we observed a complex trade-off between choosing parameters that improve short response times (0-500ms) versus parameters that reduce the number of flows with long response times (greater than 1000ms). We have chosen to favor those parameter settings that improve performance for the largest fraction of flows, and hence have focused on improving response times for the flows with short response times.

Quantitatively 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-

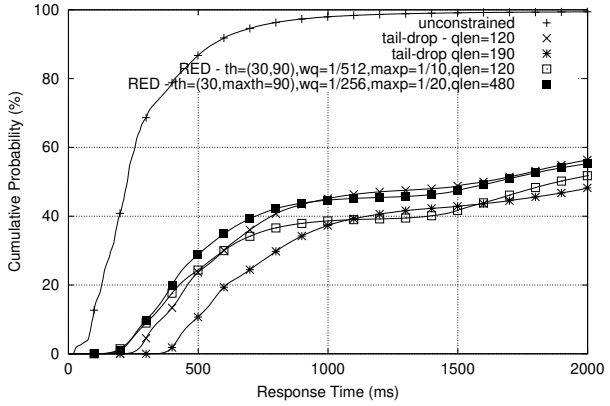
tested engineering guidelines for RED deployment and the complexity of setting RED parameters, there seems to be no advantage to RED deployment on links carrying only Web traffic.



(a) 90% load



(b) 98% load



(c) 110% load

Figure 5.12: *Tail-drop and RED comparisons at different offered load levels.*

Chapter 6

Analysis of RED Response Times

When discussing the results from evaluation of RED that we presented in chapter 5, a reoccuring question has been whether there is a single dominating cause of the observed reduction in HTTP response time performance.

To answer this question we have conducted two studies that provides insights on the impacts of congestion on the response time performance and arrival process at the bottleneck router. In the first study we analyze response times for RED queue management. The results re-enforce our understanding that response times for HTTP traffic is a complex issue involving many trade-offs. The second study considers the impact that congestion has on burstiness in the generated traffic. The results show that as the load on the bottleneck router increases towards the link capacity the TCP/IP congestion control and avoidance algorithms changes the packet arrival from a bursty too a smoothened arrival process. Essentially this change allows high bandwidth utilization while maintaining a low drop rate on the bottleneck router.

6.1 Impact of Packet Drops/Retransmissions

In this section we present a short analysis in which we document the impact of packet drops on the RED response times. The results of this analysis shows that no single class of packets cause deterioration of response time performance, it is rather a mix between many different events and thus we conclude that response times for HTTP traffic is a complex issue involving many trade-offs.

6.1.1 Methodology

For this analysis we are particularly interested in determining the general impact of packet drops on response time performance, and whether a specific class of packet drops dominate the change in response time performance one a congested network. In order to shed some light on these issues we added further instrumentation to the laboratory network in order to collect more detailed traces.

Instrumentation

Data is collected from the laboratory network by adding further instrumentation, as shown in the network diagram in Figure 6.1. We have added two 100Mbps hubs, that are placed between the switches and the routers. These hubs are then connected

to the monitoring machine which collects packet header traces (TCPdumps) for all traffic traversing the hubs during an experiment. An important point is that only one monitoring machine is used in order to ensure timestamping of packets according to a single clock.

Through postprocessing of the collected traces we link the exact packet traces for request/response pairs with the actual measurements of response times. This allow us to calculate more detailed statistics on the number of retransmitted packets and the period from when a packet has been dropped until it is retransmitted.

Introducing hubs on the 100Mbps links forces the links to run in half-duplex mode where Ethernet packet collisions are possible. A study of the `netstat` log file for an experiment at 98% offered load shows collisions for approximately 1-2% of the packets. With the retransmission times for Ethernet lying below 0.5ms for first and second retransmission on Ethernet, then this is an insignificant delay compared to the fact that the bottleneck link can only forward approximately 1 packet per ms. Thus the additional delay introduced by a collision is comparable with the insignificant delay of a packet when it arrives at a queue already holding one packet.

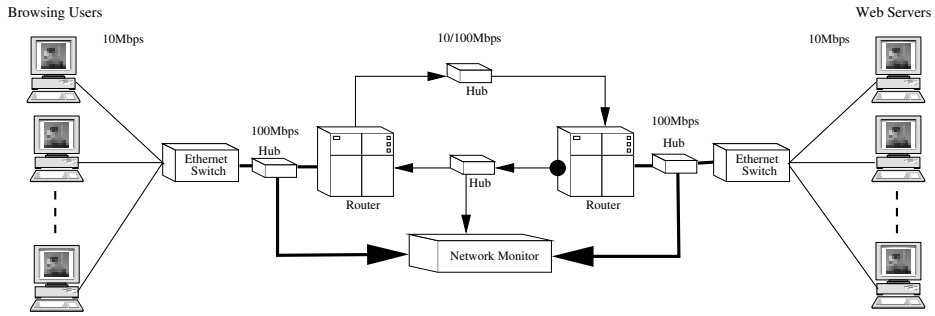


Figure 6.1: *Laboratory network with additional instrumentation.*

Experiments

For the analysis we repeat two RED experiments that previously has produced clearly different response time results. Both experiments are run at 98% offered loads and with fixed parameter settings of $max_p = 1/10, w_q = 1/512$, and $qlen = 480$. The difference between the experiments lies in the threshold parameters. These are set to $(min_{th}, max_{th}) = (5, 15)$ for the first experiment, and $(60, 180)$ for the second experiment.

6.1.2 Results

To summarize the measurements from these experiments, we have grouped the flows of each experiment into four classes dependent on retransmissions on the path from the server to the client (no retransmissions have been observed on path from the client to the server). As described in Table 6.1, the first class describe flows with no retransmissions, the next classes describe flows that have experienced one or more retransmissions of SYN,FIN, or data packets, while the last class describe flows with any combination of SYN, FIN or data retransmissions. The class of FIN retransmissions is relevant since these are often piggy-bagged on the final data packet of a transmission, therefore a lost FIN will influence the measured response time.

Table 6.2 and Table 6.3 summarizes the results from the experiments. From Table 6.2 we can see that there is a clear difference in retransmission characteristics for the two experiments. In the experiment with $(min_{th}, max_{th}) = (5, 15)$, we see that 56.1% of all the flows have no retransmissions, while in the experiment with the larger thresholds it is 87.1% of the flows that have no retransmissions. Looking at the distribution between the different classes of retransmissions it is clear that data-packet retransmissions is the dominant class, but SYN and FIN retransmissions play a significant role.

Class	Description
NR	Flows with no retransmissions
S+	Flows with one or more SYN retransmissions
F+	Flows with one or more FIN retransmissions
D+	Flows with one or more data retransmissions
SFD+	Flows with any combination of SYN, FIN, or data retransmissions

Table 6.1: Retransmission classes.

Class of retransmission event	% of all requests	
Experiment (min_{th}, max_{th})	(5,15)	(60,180)
NR	56.1	87.1
S+	7.4	2.0
F+	6.0	2.0
D+	25.5	8.5
SFD+	5.0	0.4
Total 1 or more retransmissions	43.9	12.9

Table 6.2: *Summary retransmission statistics.*

min_{th}	max_{th}	% drops	% unforced drops	% forced drops	μ qlen	η response time	connections
5	15	12.4	69.1	30.9	11.1	498	$440 \cdot 10^3$
60	180	2.40	100	0.0	65.7	381	$460 \cdot 10^3$

Table 6.3: *Performance summary for the repeated experiments.*

Figure 6.2 gives the cumulative distributions of response times for each class of retransmission events. Also shown is the cumulative distribution of response times for all the flows (the curve in bold). Notice that we here show requests with a response time up to 6s.

From the figure we see that the response times for about 50% of the flows with FIN or data retransmissions are shifted relative to those with no retransmissions by an amount corresponding to typical retransmission timeouts in our experiments ($\approx 1.5s$). The response times for flows 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 Figures 6.2a and 6.2b we observe that the response times for those connections having retransmissions are longer in Figure 6.2b by a factor somewhat greater than the additional mean queueing delay for this case (about 65ms). Our preliminary analysis indicates that changes in the response times because of retransmissions are a complex combination of factors that influence the retransmission delays. These include:

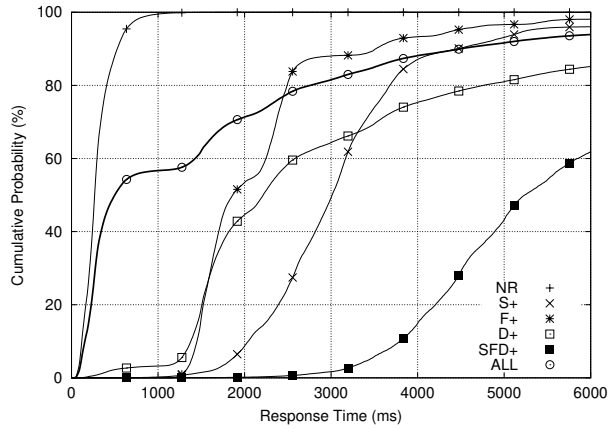
- the mean queuing delay, which influences the estimated RTT,
- the deviations in RTT caused by increased variance in queueing delays, which are magnified by a factor of 4 in the TCP algorithm for computing timeout,
- the timer granularity (500ms), and
- the minimum timeout value (1s).

See [1] for a more comprehensive analysis of these and other factors affecting TCP retransmissions.

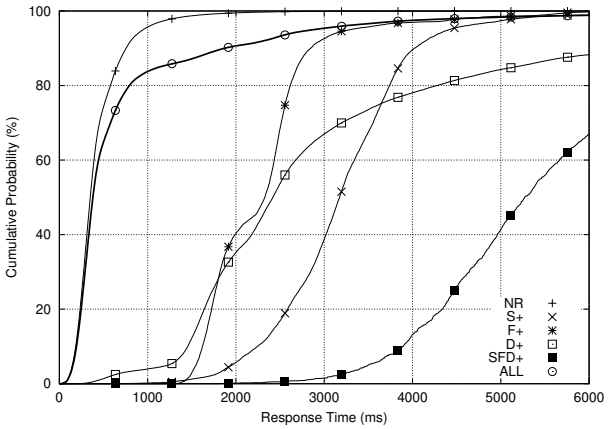
The relative contribution of each class of retransmission to the overall response time distribution is shown in the plots in Figure 6.3. We do this by showing each separate class combined with the class of flows that did not experience retransmissions. That is: those with no retransmissions, those with 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 50^{th} 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 the 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 6.4 that gives a scatter plot of response times versus server reply sizes. There is one dot in this plot for each of the approximately $400,000$ request response pairs with reply sizes less than 16KB in the experiment with $(min_{th}, max_{th}) = (5, 15)$. 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 number of replies of size less than 4KB 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 retransmissions.
- The sharp step increase in response times with no retransmissions for those responses with lengths greater than 2.88KB (corresponding to the initial TCP congestion window).
- 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).



(a) $(min_{th}, max_{th}) = (5, 15)$



(b) $(min_{th}, max_{th}) = (60, 180)$

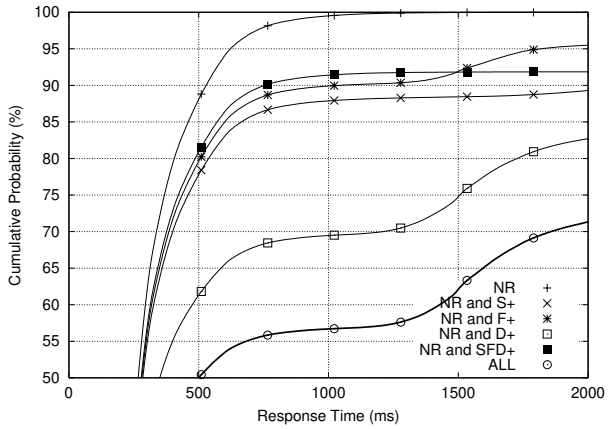
Figure 6.2: *Absolute performance of flows experiencing retransmissions.*

6.1.3 Summary

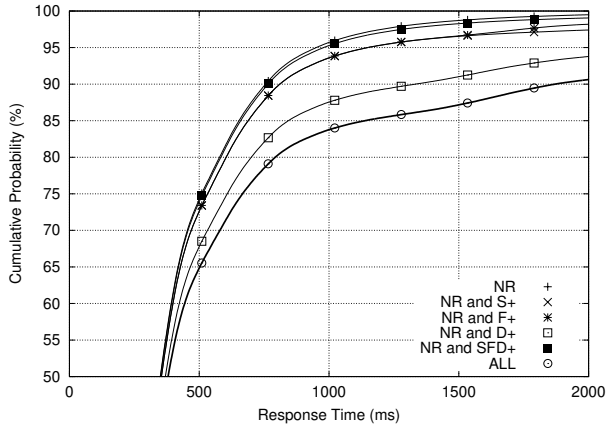
This brief analysis has re-enforced our view that understanding the effects of RED and tail-drop queue management on the end-to-end response times for HTTP traffic is a complex issue. We show that there is no single dominating cause of reduction in HTTP response times, instead we have shown that the reduction in response times involves many tradeoffs and parameters including not only parameters set on routers but also those controlled at the end-systems (e.g., TCP retransmission parameters).

6.2 Congestion and Web-like Traffic

In our second study we analyze effect of a bottleneck link on the behavior of the generated traffic. The results show, that even low levels of congestion influences the traffic behavior by smoothening the packet arrival process at the bottleneck router.



(a) $(min_{th}, max_{th}) = (5, 15)$



(b) $(min_{th}, max_{th}) = (60, 180)$

Figure 6.3: *Relative contribution of flows experiencing retransmissions to total distribution.*

We argue that the smoothening process is a consequence of the TCP congestion control algorithms, however, determining the exact process of the smoothening requires additional work.

6.2.1 Methodology

For these experiments the instrumentation of the laboratory network described in the Section 6.1 is reused. By capturing header traces of packets on the network segment before the bottleneck router, we are able to monitor the packet arrival process at the bottleneck router.

To get an overview of the impact of the packet-arrival process with a bottleneck router we ran a series of experiments with RED queue management at different offered loads. Starting at 50% offered load where no packets were dropped by the bottleneck router, and increasing it to 70, 80, 90, and ending at 98%.

The parameter setting for RED is shown in Table 6.4, and remained fixed for the

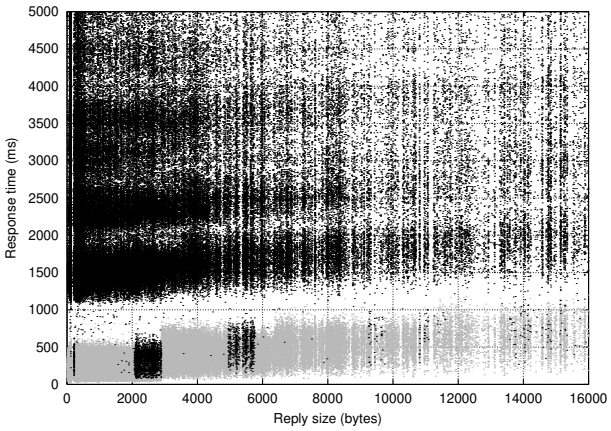


Figure 6.4: *Scatter plot of response times versus reply size under RED for* $(min_{th}, max_{th}) = (5, 15)$. *Flows with no retransmissions are marked with gray dots, while flows with retransmissions are marked with black dots.*

experiments used in this section. These correspond to our choice of recommended parameters that was determined in the analysis of REDs impact on HTTP response times, see Chapter 5 on page 59.

Parameter	Value
min_{th}	30
max_{th}	90
w_q	1/512
max_p	1/10
qlen	480

Table 6.4: *RED parameter settings.*

6.2.2 Results

Table 6.5 gives the summary statistics for the 5 experiments that we have performed for this analysis. From this we can see that the experiments behaved as we expect them to, based on our knowledge from previous experiments. In particular we observe that packet drops grow from 0.01% at 50% load to 4.3% at 98% load, and that the average queue length grow from 1.4 packets at 50% load to an average of 40 packets at 98% load.

Figure 6.5 shows the packet arrival process on the network ahead of the bottleneck router for loads ranging from 50-90% offered load. Each plot shows the number of packets arriving on the network from the "Web servers" per second.

What we find interesting is that the arrival process is smoothened significantly, as the load on the network is increased. At 50% we have a bursty arrival process that is similar to the behavior which we have observed on the unconstrained network. If we compare the arrival process at 50% load and the process at 80% offered load, then we see a change in the general behavior of the arrival process. At 80% offered load, bursts are wider and there appears to be a common maximum burst size 1300

% load	% drops	% unforced drops	% forced drops	μ qlen	max qlen	$\frac{KB}{s}$
50	0.01	100	0.0	1.43	86	629
70	0.10	100	0.0	5.62	100	835
80	0.26	100	0.0	10.9	123	967
90	1.04	100	0.0	21.7	119	1091
98	4.30	99.5	0.53	39.8	149	1162

Table 6.5: *Performance summary for the repeated experiments.*

packets/s. At 90% offered load this tendency is even more clear, and the arrival process can be described as smooth, rather than bursty, with an arrival rate around 1300 packets/s. At 98% (see Figure 6.6), the behavior is similar to what we have seen at 90% offered load.

To study the change in behavior of the arrival process we take the analysis a step further by performing the test of fractal like behavior that we also used in Section 3.5.2 on page 41. Figure 6.6 shows the result of this test for the experiment running at 98% offered load on the constrained network and as reference Figure 6.7 shows the result from an experiment running at 98% load on the unconstrained network. The marked intervals correspond to the range covered by "zoom" plot with a finer Time Unit.

Comparing Figure 6.6 and 6.6 we clearly see that there is a significant difference between the arrival processes of the two experiments. In particular we see no bursts of traffic in the plots from the constrained network when averaging over intervals of 1 and 0.1 seconds. However, when averaging in intervals 0.01s both experiments remain to have a bursty arrival process. In general we conclude that the behavior of the packet arrival process at 98% offered load on the constrained network is not fractal-like, because plots qualitatively change dependent on the timescale.

We find that the smoothened arrival process is a natural consequence of introducing a bottleneck link in the experimental network, rather then being an artifact. In the following we present analyze our results and argue for why the smoothening of the arrival process occurs.

The basic observation is that increasing the traffic load towards the link capacity changes two aspects:

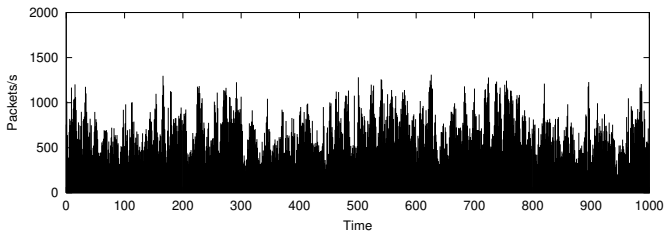
- The average queue size on the bottleneck router increases; more packets are buffered before reaching their destination, and
- packets are dropped.

In the following we analyze how these aspects influence the packet arrival process.

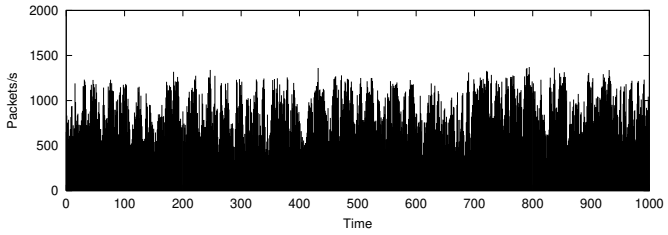
Impact of buffering

When the bottleneck link is used, packets are stored in the buffer for a period, and then forwarded when bandwidth becomes available. As a consequence traffic arriving to the non-empty buffer will be smoothened when forwarded. This smoothening has considerable influence on the behavior of TCP senders in our experiments because of the self-clocking behavior imposed by the use of the sliding window algorithm in TCP.

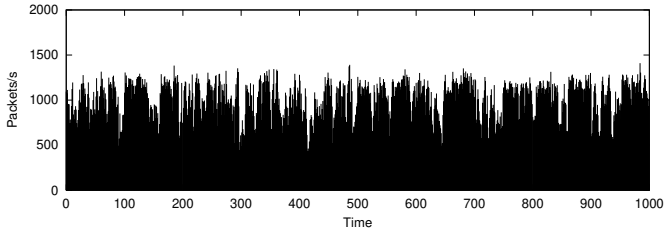
Once a sender has sent the data corresponding to the size of its current transmission window, new packets can only be sent when earlier packets have been acknowledged. Acknowledgements (ACKs) are returned by the receiver when packets are received, and thus the stream of ACKs corresponds to the arrival rate of packets at



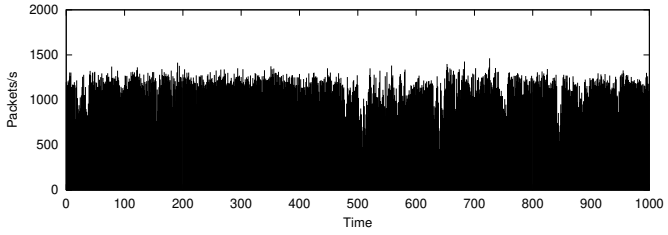
(a) 50% load



(b) 70% load



(c) 80% load

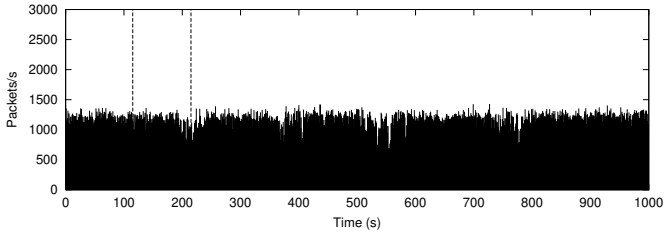


(d) 90% load

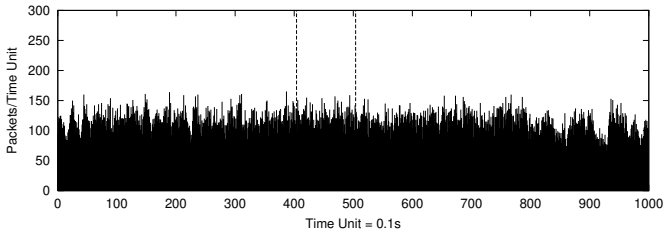
Figure 6.5: *Packet arrival process from loads ranging from 50-90%.*

the receiver. Therefore a sender can only send data as fast as the receiver receives it or increases the size of the transmission window.

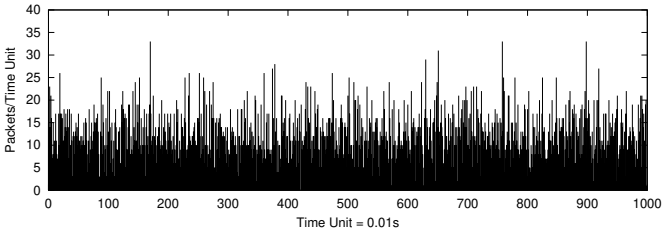
In our case, the cumulative bandwidth of data that all receivers can accept at any time is 10Mbps, and thus on average only 10Mbps of data can be acknowledged. Consequently traffic generators have limited room for generating bursts of traffic.



(a) Time Unit = 1s, marked interval from $115 - 215$s.



(b) Time Unit = 0.1s, marked interval $40.4 - 50.4$s.



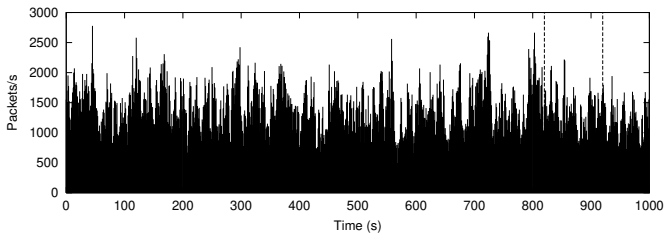
(c) Time Unit = 0.01s

Figure 6.6: *Behavior on the constrained network at 98% offered load.*

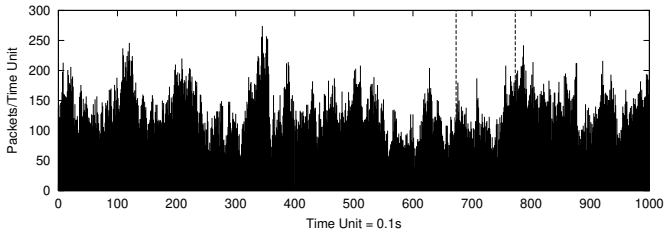
In fact creating new connections and increasing the size of transmission windows is the only way to produce a burst of traffic, and TCP congestion control algorithms significantly limits the potential size of these bursts.

ACK-compression [61, 71] has been described as a potential cause of bursty packet arrival processes. It occurs in networks where congestion arises on the transmission path on which acknowledgements travel. An example scenario is that a receiver responds to receiving data packets by sending a stream of ACKs corresponding to a data packet arrival rate of 10Mbps arrive at a non-empty router queue. Since ACK-packets are sent in response to arriving data packets, the density of the ACK-packets is what characterizes the arrival rate at the receiver. However, ACK-packets are much smaller than data packets (40bytes versus 1KB), therefore, when a sequence of ACK packets are queued and then later forwarded from a queue, then the density of the packets may have been significantly increased compared to when they arrived. This causes the ACK-stream to correspond to a much higher arrival rate than before being buffered.

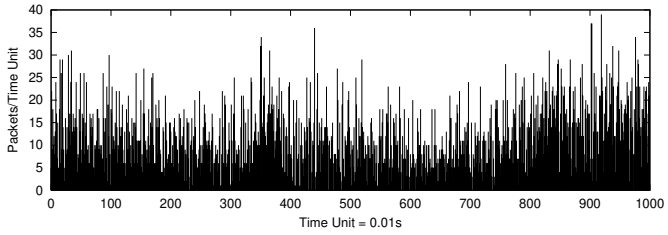
We have not included elements of ACK-compression in our evaluation of RED



(a) Time Unit = 1s, marked interval from 820 − 920s



(b) Time Unit = 0.1s, marked interval from 67.3 − 77.3s



(c) Time Unit = 0.01s

Figure 6.7: *Behavior on the unconstrained network at 9.8Mbps load.*

queue management and it is difficult to say how this may effect the results. Research has been conducted to determine role of ACK-compression on the Internet today. For example Paxson [56] concludes that it has no real effect on network performance, and Feldmann *et al.* [19] notes that ACK-compression may not have a direct impact on performance, but suggest that it may be related to the highly complex behavior of measured IP traffic. However, further research is necessary to better understand these issues.

With respect to the used traffic model, we should consider the impact of buffering, since the arrival process does not behave as predicted. By introducing a bottleneck, a fundamental problem arises, because flows can influence each others transmission rate. This essentially means that flows are no longer independent. Using the ON/OFF model we assume that ON and OFF periods are independent, however, with a bottleneck, the duration of a flow depends on other flows that are active in the same period, and thus we cannot assume independence.

Impact of packet drops

As with buffering, packet drops may also contribute to smoothening of the arrival process. For instance, one or more packet drops during a burst in the arrival process may quickly limit the size of the burst, and instead increase the period with high load by retransmitting shortly after the burst.

Again, introducing a bottleneck link may change the assumptions on which we generate traffic. A key element in generating fractal-like traffic is that the tails of the ON or OFF distributions of the traffic model are modeled well by a heavy-tailed distribution. Furthermore, the ON and OFF distributions should be independent and identically distributed, see Section 3.2.3 on page 29, and the model assumes traffic to be transmitted at a constant bit-rate.

Assuming constant bit-rate transmission may be problematic in an environment where a larger percentage of flows experience passive periods of 1-3 seconds while waiting for a retransmission timeout. With the presence of these long inactive periods Web object transfers may not be well modeled as one long ON period. Instead it may be necessary to model a Web-object transfer as a number of smaller ON periods.

Breaking Web object transfers into a number of smaller transfers again has the causes the ON/OFF distributions to change as a function of the number of packet drops. Consequently we cannot assume identically distributed ON- and OFF-periods. Furthermore, the heavy-tailed distributions may not remain heavy-tailed.

In total, if we cannot assume a transfer to be one long ON-period even with the presence of packet drops, then we cannot expect fractal-like traffic behavior.

6.2.3 Summary

We have presented an analysis of the packet arrival process with the presence of a bottleneck link. The analysis shows that limiting the bandwidth also limits the burstiness of the arrival process. Our conjecture is, that the main cause of this is the self-clocking behavior of TCP, and potentially also behavior related to drops and retransmissions.

To explain how it is possible that the general behavior of the packet arrival process changes as loads approach the capacity of the bottleneck link we look closer at the assumptions made when using the ON/OFF model to generate fractal-like traffic. The main observation is that flows are no longer independent with the presence of a bottleneck link. E.g. the bandwidth usage of one flow has an impact on the bandwidth usage of another flow. Furthermore the long passive periods of flows occurring when packets are dropped may also change assumptions about the distributions in the traffic model, thus explaining a change in the general behavior.

We also considered whether ACK-compression could potentially change the behavior of the smoothened arrival process to becoming more bursty or fractal-like. We found that there is no clear indication of this in the literature, however we do find that ACK-compression should considered included in a future evaluation model, however, a model, describing the general behavior and presence of ACK-compression on the Internet, is necessary.

Overall this analysis provides an explanation to why it is possible up to 80-90% of the bottleneck link capacity without seeing significant deterioration in HTTP response time performance.

6.3 Summary

In this chapter we have presented two studies that describe the lower level changes in traffic behavior as the load in our experimental network increases towards congestion. In particular we describe the impact of packet drops and retransmissions on the response time performance and changes in the packet arrival process as the load on the bottleneck router increases.

The primary motivation for studying the impact of packet drops and retransmissions was to determine whether there is a single dominating cause of the observed reduction in HTTP response times. From the study we learn that the changes in response times as a result of packets drops and retransmissions is a complex issue. It involves many tradeoffs, including not only parameters set on routers but also those controlled at the end-systems (e.g., TCP retransmission parameters).

In the second study we show that the packet arrival process at the bottleneck router significantly changes as the load on laboratory network increases towards congestion. As a result we have observed that at 80% and higher offered load the packet arrival process is smoothened. We argue that the cause of the cause of this change is due to the TCP congestion control and avoidance mechanisms rather then an artifact occurring in the laboratory network. Additionally we consider the potential impact of modeling ACK-compression in the experiments. We find that this should be considered in a future experiments, however, a model, describing the general behavior and presence of ACK-compression on the Internet, is necessary.

Chapter 7

Conclusion and Further Work

In RFC (Request For Comments) 2309 [9] the active queuing mechanism RED was purposed for widespread deployment on Internet routers. This dissertation is a comment; we have performed an empirical evaluation of RED using a laboratory network in which Web-like traffic was generated using a well founded and widely accepted HTTP traffic model. The performance of RED was measured in terms of HTTP response times, and compared to the performance of traditional tail-drop (FIFO) queuing.

7.1 Results

The results of the experiments with RED queue management are summarized as follows:

For offered loads near or below the levels of link saturation (90% or less), the response times are fairly close to what we would experience on an unconstrained network without congestion. Additionally, at these loads we see little difference in the response times between the best tuned RED and the best tuned tail-drop queuing 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. Worse, we show that one can use plausible values for certain RED parameters and get 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 queuing. However, this difference is likely only to be significant between 90% and 100% offered loads. The reason is that at loads above 100%, the performance degrades so rapidly that any improvement is likely to be a second-order effect. Additionally, between 90-100% offered load, the 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. This is important because the parameters that out performed tail-drop queuing were found only through extensive trial an error experimentation. Furthermore, it is also the case that the RED parameters that provide the best link utilization or lowest drop rate at this load, produce poorer response times.

Overall the study of RED has provided a general insight on the behavior of HTTP response times. First, our study shows that there is a complex trade-off between choosing parameters that improve short response times (0-500ms) versus parameters that reduce the number of flows with long response times (greater than

1000ms). We have chosen to favor those parameter settings that improve performance for the largest fraction of flows, and hence have focused on improving response times for the flows with short response times. Secondly, the study demonstrates that optimizing for highest link utilization and/or lowest loss rates has a negative effect on response time performance.

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 some reason, Web-dominated traffic is 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 benefit to be gained by RED deployment on links mainly carrying Web traffic.

7.2 Further Work

The methodology we have used for evaluating REDs performance is limited in two ways. We only consider congestion on the path from servers to clients and we only model Web-like traffic in the network.

As we discussed in Section 6.2 on page 77, congestion on both paths may introduce some changes in the behavior of the traffic. ACK-compression is a potential problem for performance in such an environment. However, to increase validity of such a study we need better models that describe ACK-compression and maybe also congestion on the Internet. To our knowledge such models are currently not available.

We study 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. Especially the introduction of UDP-based traffic may show interesting results, since the behavior of this traffic deviates significantly from TCP traffic by not reacting to congestion notifications. Models of UDP traffic have been developed making it possible to attack this question.

Many new transfer technologies have been proposed for improving the HTTP response times on the Internet. Most significant is probably the introduction of HTTP/1.1 [22] which among other introduces the idea of persistent connections, see [48], which allows pipelining of several Web objects over a single TCP connection. Evidently, this minimization of the overhead is likely to improve HTTP response times, as shown by Nielsen *et al.* [52]. It is unclear how HTTP 1.1 will change the general traffic behavior. We find it relevant to persue a study of the impact of using RED in combination with a model of HTTP 1.1 traffic. However, if the new model has characteristics similar to the Mah model, we believe that the performance impact will be quite similar to what we have observed in our study.

A second technology that may improve HTTP response times is Explicit Congestion Notification (ECN) [24]. ECN changes the TCP congestion notification mechanism from being implicit, signalled through packet drops, to being explicit by setting bits in packet headers. For instance a router using RED queue management will set a bit in the packet header instead of dropping a packet early. ECN may therefore limit the number of packet drops. Using ECN in combination with RED will essentially be reduced to problem quite similar to what we have experienced with tuning RED: Making a choice of parameters avoiding large queues while avoiding underutilized links. One should consider evaluating ECN and RED to quantify the parameter range in which RED operates well with ECN.

Our experiments are based on an HTTP traffic model that is derived from

network traces taken in 1995. This may question the validity of the current model. On the other hand Bruce Mah has analyzed the distribution of HTTP response sizes and hereby provided evidence that this distribution may be heavy-tailed. This links the behavior found in the Bruce Mah's model with more recent results showing heavy-tailed distributions of Web reply sizes. If more recent models shows significant changes in the behavior of Web traffic, these should be applied in a similar evaluation of RED queue management. Such new models are currently being developed at University of North Carolina [62].

In summary, we find that extending the evaluation method to include aspects of ACK-compression and modeling a mix between different classes of traffic to be of greatest interest, because these aspects could change the nature of the packet arrival process on the bottleneck router.

7.3 Summary

To answer RFC 2309 [9]. In an environment dominated by Web traffic we have found that RED provides no advantage over tail-drop queuing when considering end-to-end response times. Further work in performance analysis is necessary in order to extend the evaluation methodology. Most interesting is addressing the problems of including other classes of traffic in the evaluation, and study the impact of ACK-compression.

Appendix A

Laboratory Network

A.1 Network Diagram

Figure A.1 shows a detailed diagram of the laboratory network. Table A.1 lists machines and their hardware configuration. Network cards on end hosts are mainly either Intel EtherExpress Pro or 3COM-905 cards. On the congested interface on daffy we use the 3COM-905 for 10Mbps experiments and an Intel EtherExpress Pro for experiments on the unconstrained network.

The Switch we use is a Cisco Catalyst 5000. The hub's are standard 10Mbps or 100Mbps hub's

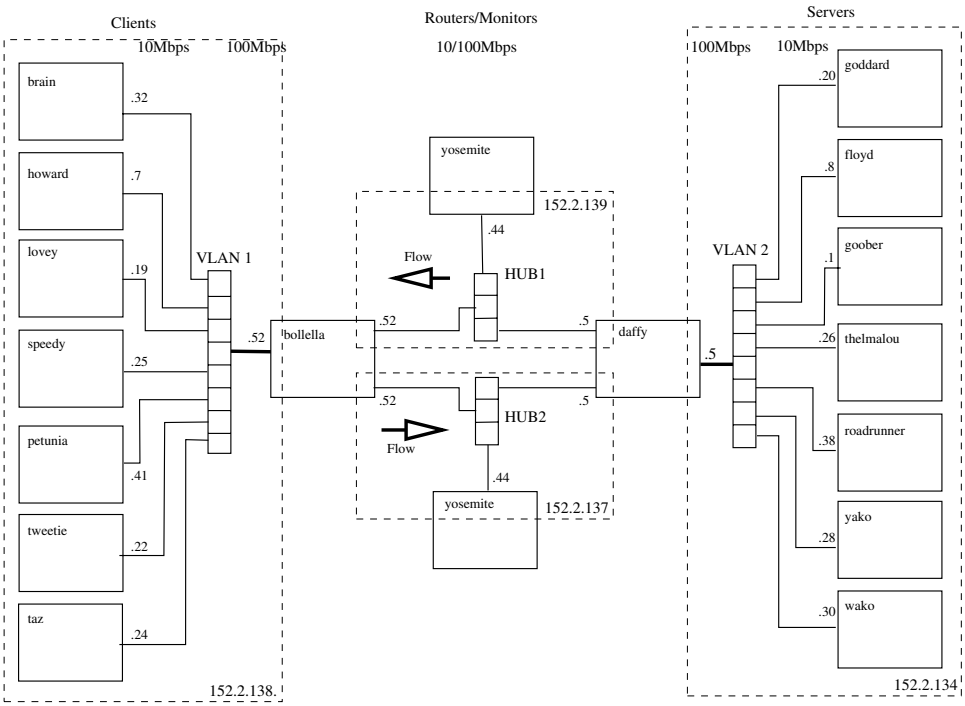


Figure A.1: *Detailed diagram of laboratory network.*

Hostname	CPU	MEM (MB)
brain	Pentium II 400MHz	128
howard	486DX 66MHz	16
lovey	468DX 66MHz	16
speedy	Pentium 120MHz	32
petunia	Pentium II 400MHz	128
tweetie	Pentium II 450MHz	256
taz	Pentium II 450MHz	256
goddard	Pentium 90MHz	16
floyd	486DX 66MHz	16
goober	486DX 66MHz	16
thelmalou	Pentium 120MHz	32
roadrunner	Pentium II 400MHz	128
yako	Pentium II 300MHz	64
wako	Pentium II 450MHz	256
bollella	Pentium II 300MHz	64
daffy	Pentium II 300MHz	64
yosemite	Pentium II 400MHz	128

Table A.1: *Machine Configurations.*

Appendix B

Experiments with Tail-drop and RED

This appendix provides additional plots and statistics for the experiments on which the RED versus tail-drop comparison, Chapter 5 on page 59, was done. Some of the plots shown here are identical to the ones shown in the presentation of the results, however we have included these to provide a complete picture of the experimental results. First we give complete listings of the experiment configurations tried for each of the queuing mechanisms, then we show plots and summary statistics for each of these.

Throughout this presentation of results a number of abbrevations are used. These are defined in table B.1.

Abbreviation	Meaning
bwdp	bandwidth delay product
i_1	% of response times $\leq 1s$
i_2	% of response times from $1 - 2s$
i_3	% of response times from $2 - 3s$
i_4	% of response times greater than $3s$
η rsp-time	median response time
μ *qlen*	mean queue length

Table B.1: *Abbreviations used in the presentation of results.*

B.1 Experiments

B.1.1 Tail-drop Experiments

Table B.2 shows the combination of parameters that we have tested with tail-drop queuing on the constrained network.

B.1.2 RED Experiments

Table B.3 gives an overview of the experiments performed with RED queuing on the constrained network. Each row in the table describes a subset of experiments where all combinations of the given parameters are tested.

qlen \ load	50	70	80	90	94	98	102	110	130
15	0	5	10	15		20		25	30
30	1	6	11	16		21		26	31
60	2	7	12	17	43	22/56/57	46	27	32
90				40	44	41	47	42	
120	3	8	13	18	45	23/58/59	48	28	33
150				49		50		51	
190			60	61		62		63	
240	4	9	14	19		24		29	34
1000				52		53		54	

Table B.2: *Tail-drop experiments, constrained 10 Mbps network. Load is the offered load in percent and qlen is the size of the queue in terms of packets. Each cell contains on or more* id *each corresponding to an experiment.*

B.2 Tail-drop Summary

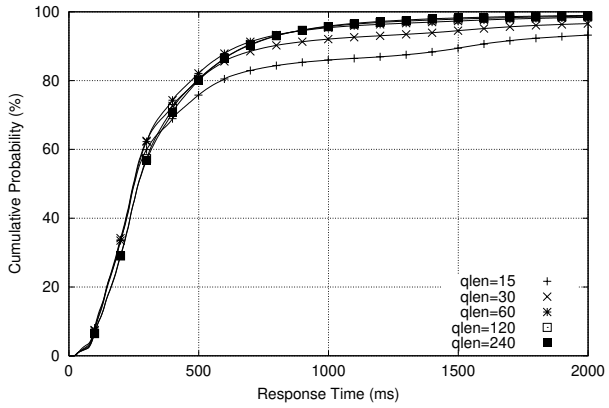
The tail-drop results are presented in two series of plots. The first series illustrates the effect of changing the queue length under different loads. The second series shows the effect of changing the load with a given queue length.

The effect of changing the queue length under different offered load levels is shown in Figure B.1 and B.2. Table B.4 gives summary statistics for these plots.

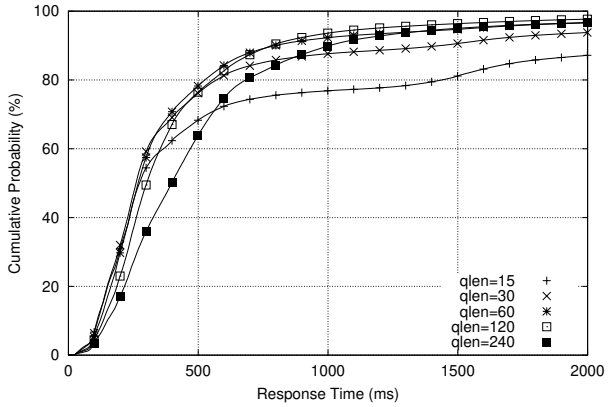
The second tail-drop series, Figure B.3 and Table B.5, shows the effect of increasing the offered load from 50% to 110% at two different queue lengths. The point here is that the overall effect is the same if we choose a queue length of 60 or 120 packets. Table B.5 gives summary statistics for the experiments in the plots.

Purpose	min_{th}	max_{th}	w_q	$maxp$	$qlen$	load
Test min_{th}, max_{th}	5, 15, 30, 60, 120	$3 \cdot min_{th}$	1/512	1/10	480	50, 70, 80, 90, 98, 110, 130, 150
Test default setting	5	15	1/512	1/20	60	50, 70, 80, 90, 98, 110
Test w_q, max_p, max_{th}	5	90, 120, 150	1/512, 1/256, 1/128	1/20, 1/10, 1/4	480	90, 98, 110
Test w_q, max_p	30	90	1/512, 1/256, 1/128	1/20, 1/10, 1/4	480	90, 98, 110
Test different loads	5	90	1/512, 1/256, 1/128	1/20, 1/10	480	94, 102
Test min_{th}	30, 60, 90	150	1/512	1/10	480	90, 98, 110
Test min_{th}	15, 45, 60	90	1/512	1/10	480	90, 98, 110
Test max_{th}	60	90, 120, 150, 180, 240	1/512	1/10	480	90, 98, 110
Test max_{th}	5	45, 180, 150, 240	1/512	1/10	480	90, 98, 110
Test max_{th}	30	45, 60, 180	1/512	1/10	480	90, 98, 110
Test $qlen$	60	150	1/512	1/10	120, 160, 240	90, 98, 110
Test $qlen$	30	90	1/512	1/10	120, 160	90, 98, 110

Table B.3: *List of RED experiments on the constrained 10Mbps network. Each row corresponds to a series of experiments where all combinations of the given parameters are tested.*

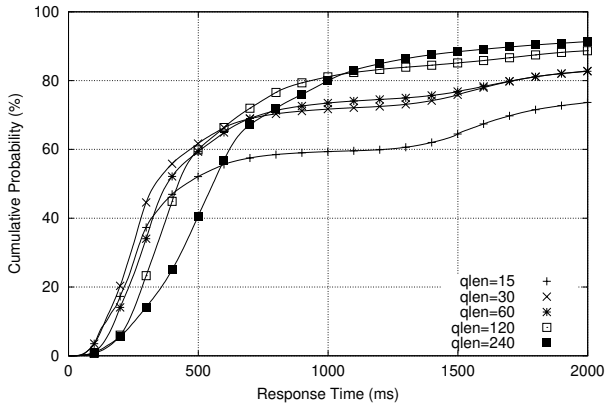


(a) 80% load

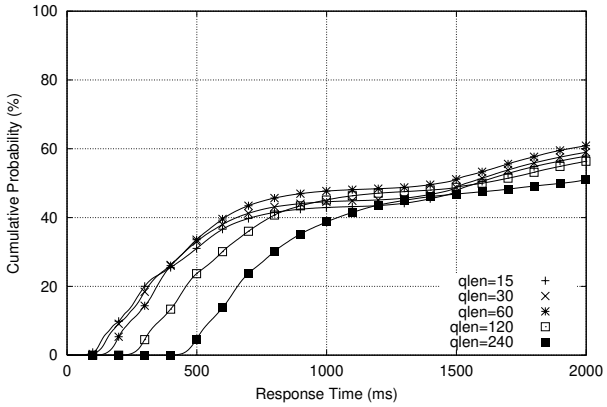


(b) 90% load

Figure B.1: *Queue length variation, 80-90% load.*



(a) 98% load



(b) 110% load

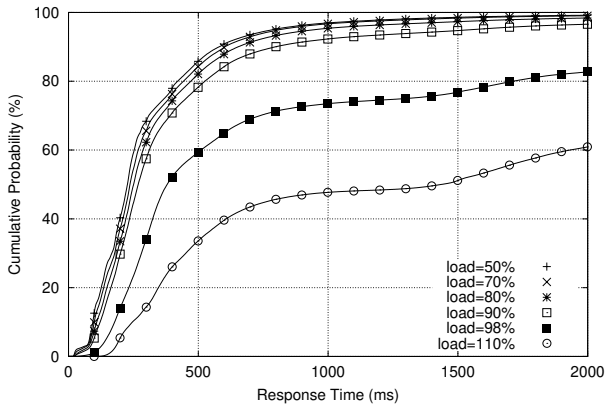
Figure B.2: *Queue length variation, 98-110% load.*

load (%)	*qlen*	$\frac{KB}{s}$	% drops	$\mu qlen$	η rsp-time (ms)	i_1	i_2	i_3	i_4
80	15	987	2.60	3.20	253	85.9	7.29	4.22	2.59
80	30	992	1.11	6.48	249	92.1	4.50	2.14	1.28
80	60	980	0.32	11.7	259	95.2	3.15	0.95	0.68
80	120	990	0.12	22.7	268	95.7	3.02	0.64	0.65
80	240	981	0.00	25.8	269	95.8	3.14	0.50	0.54
90	15	1059	5.19	4.69	274	76.9	10.3	6.74	6.08
90	30	1081	2.20	9.94	261	87.6	6.17	3.75	2.48
90	60	1075	0.92	20.0	268	92.3	4.25	2.05	1.35
90	120	1085	0.34	40.3	302	93.6	4.05	1.28	1.04
90	240	1101	0.29	85.7	398	89.8	6.88	1.56	1.70
98	15	1128	11.4	7.13	455	59.4	14.3	10.2	16.2
98	30	1163	6.69	16.6	336	71.7	11.1	8.53	8.64
98	60	1177	6.22	41.6	443	72.9	9.19	7.98	9.92
98	120	1178	3.07	84.8	423	81.1	7.58	5.16	6.13
98	240	1167	1.43	154	558	79.8	11.5	3.66	5.01
110	15	1154	20.2	9.07	1583	42.9	15.0	11.0	31.2
110	30	1183	18.9	22.6	1541	44.6	14.4	11.2	29.9
110	60	1188	16.4	52.4	1582	46.9	12.5	12.0	28.6
110	120	1188	17.0	112	1601	45.1	11.3	13.0	30.6
110	240	1188	16.5	232	1921	38.6	12.3	13.9	35.1

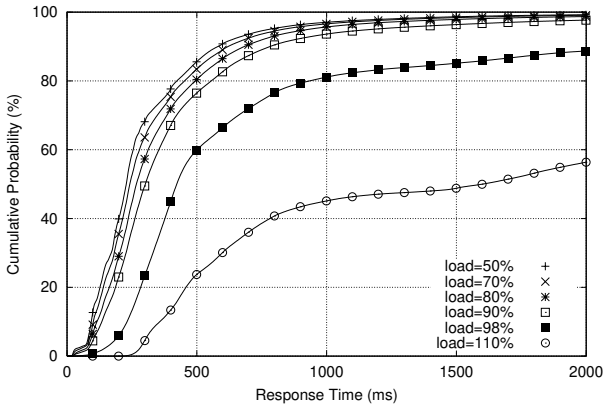
Table B.4: *Summary statistics for the experiments at different offered loads.*

load (%)	*qlen*	$\frac{KB}{s}$	% drops	$\mu qlen$	η rsp-time (ms)	i_1	i_2	i_3	i_4
50	60	631	0.01	1.43	227	97.0	2.23	0.36	0.45
70	60	860	0.10	6.05	237	96.7	2.33	0.47	0.50
80	60	980	0.32	11.7	259	95.2	3.15	0.95	0.68
90	60	1075	0.92	20.0	268	92.3	4.25	2.05	1.35
98	60	1177	6.22	41.6	443	72.9	9.19	7.98	9.92
110	60	1188	16.4	52.4	1582	46.9	12.5	12.0	28.6
50	120	615	0	1.40	221	97.2	2.05	0.31	0.44
70	120	879	0.02	11.2	243	96.6	2.51	0.41	0.50
80	120	990	0.12	22.7	268	95.7	3.02	0.64	0.65
90	120	1085	0.34	40.3	302	93.6	4.05	1.28	1.04
98	120	1178	3.07	84.8	423	81.1	7.58	5.16	6.13
110	120	1188	17.0	112	1601	45.1	11.3	13.0	30.6

Table B.5: *Summary statistics for the experiments with where the queue length is varied.*



(a) qlen = 60



(b) qlen = 120

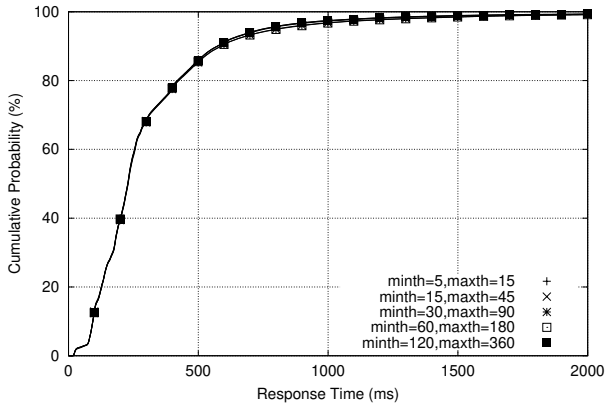
Figure B.3: *Queue length plots.*

B.3 RED summary

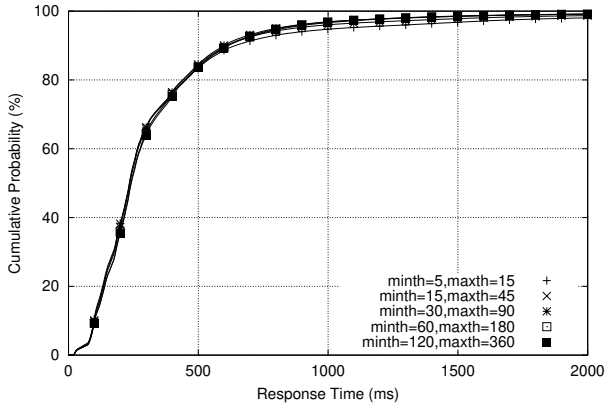
In the following sections we present the series of experiments done for the evaluation. Each series is presented with a plot and a table with summary statistics covering the experiments in the plot. We start out with the results of experiments where the thresholds are changed, then we present results evaluating the effect of each of the threshold parameters separately. Then we show the results of studying the effect of w_q and max_p. Finally we study the effect of reducing the queue length.

B.3.1 Experiments with min_{th} and max_{th}

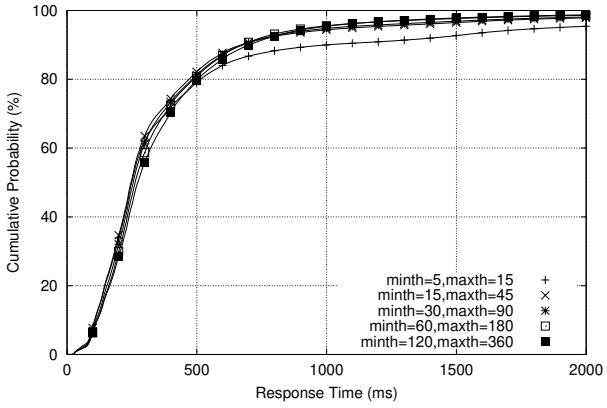
The effect of the threshold values were studied by first running a series of experiments with different min_{th} values while keeping $max_{th} = 3min_{th}$. Figures B.4 and B.5 show the results of changing the threshold values at different load levels ranging from 50 to 110% offered load. The summary statistics for the experiments in these plots are listed in Table B.6.



(a) load = 50%

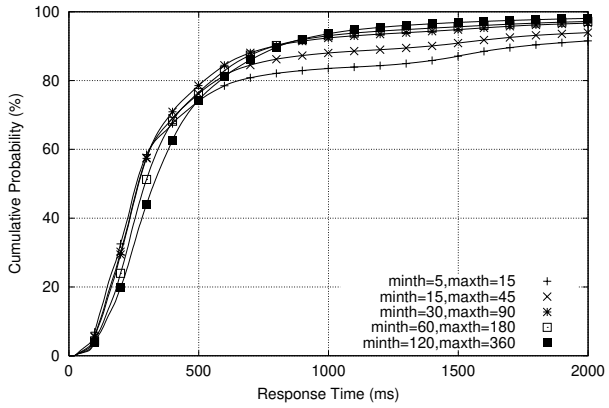


(b) load = 70%

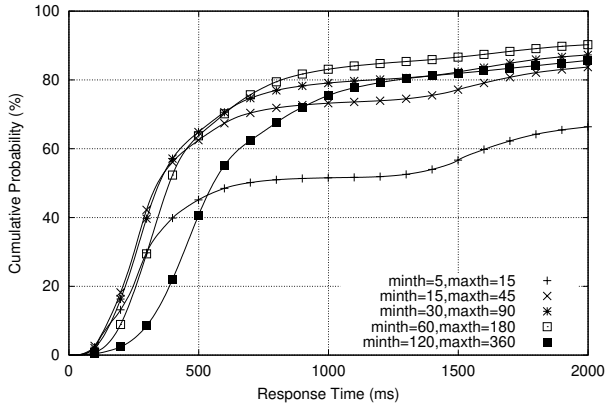


(c) load = 80%

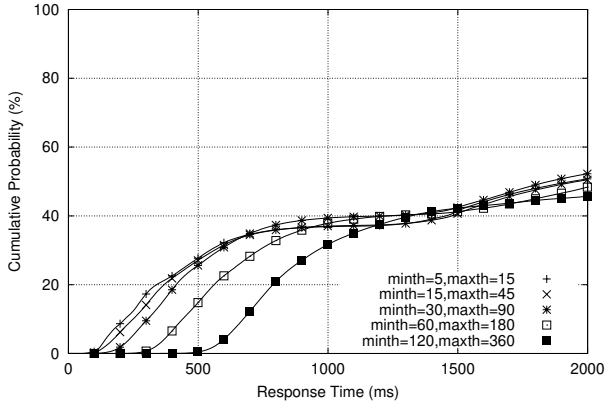
Figure B.4: *Threshold experiments 50-80% load.*



(a) load = 90%



(b) load = 98%



(c) load = 110%

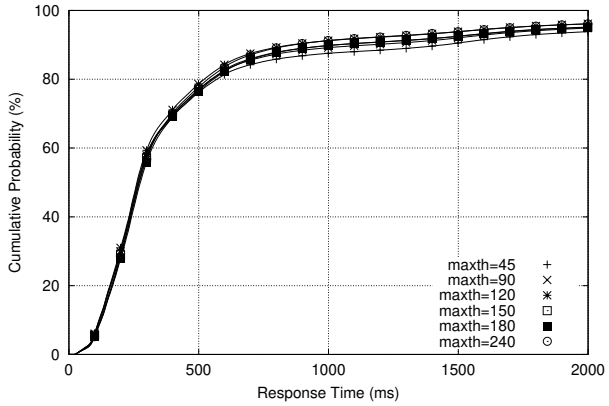
Figure B.5: *Threshold plots 90-110% load.*

load (%)	min_{th}	max_{th}	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
50	5	15	601	0.08	0.83	228	97.3	1.96	0.36	0.40
50	15	45	613	0.03	1.25	229	97.4	1.86	0.32	0.39
50	30	90	621	0.01	1.45	230	96.8	2.36	0.39	0.44
50	60	180	625	0.00	1.65	229	96.7	2.41	0.41	0.44
50	120	360	609	0	1.58	230	97.4	1.93	0.27	0.36
70	5	15	860	0.53	2.87	236	94.7	3.32	1.14	0.80
70	15	45	868	0.24	4.74	235	96.0	2.65	0.72	0.59
70	30	90	846	0.08	5.69	237	96.9	2.19	0.44	0.46
70	60	180	852	0.03	8.44	241	96.8	2.32	0.39	0.45
70	120	360	863	0.01	10.5	242	96.7	2.48	0.39	0.47
80	5	15	991	1.55	5.08	249	90.0	5.41	2.71	1.87
80	15	45	955	0.53	7.46	247	94.4	3.47	1.28	0.87
80	30	90	991	0.38	12.8	255	94.8	3.30	1.08	0.82
80	60	180	1001	0.15	19.1	263	95.6	3.02	0.75	0.66
80	120	360	988	0.04	25.7	273	95.4	3.30	0.61	0.66
90	5	15	1068	3.15	7.11	260	83.5	8.02	4.79	3.66
90	15	45	1088	2.32	15.3	267	88.0	5.91	3.42	2.64
90	30	90	1079	0.83	20.2	269	92.4	4.30	1.98	1.32
90	60	180	1095	0.51	35.5	294	93.0	4.14	1.62	1.19
90	120	360	1094	0.14	53.8	328	93.6	4.46	1.03	0.89
98	5	15	1135	15.2	11.6	686	51.5	14.8	10.2	23.4
98	15	45	1158	5.82	24.0	342	73.2	10.5	7.91	8.34
98	30	90	1164	4.09	39.4	349	79.1	8.18	6.33	6.39
98	60	180	1178	2.42	69.1	388	83.1	7.14	5.03	4.71
98	120	360	1182	3.06	147	557	75.5	10.2	6.05	8.28
110	5	15	1147	24.0	12.6	1938	36.9	13.9	9.86	39.4
110	15	45	1175	23.4	36.1	1961	37.0	13.4	10.3	39.2
110	30	90	1187	19.7	76.0	1852	39.4	12.9	12.2	35.6
110	60	180	1187	17.9	158	2123	37.8	10.5	15.5	36.1
110	120	360	1188	15.5	303	2469	31.6	14.0	14.5	39.8

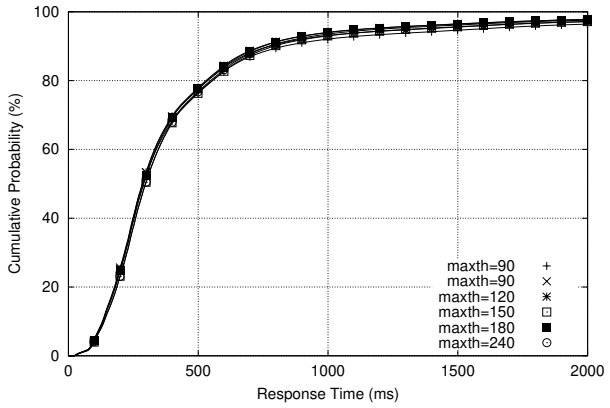
Table B.6: *Summary statistics for RED experiments where the offered load is varied while qlen* $= 480$, $max_p = 1/10$ *and* $w_q = 1/512$.

B.3.2 Experiments with max_{th}

Based on the results of experiments where max_{th} was changed as a function of min_{th} we ran a series of experiments where we only study the impact of changing max_{th}. This series was only run for offered load levels ranging from 90 to 110% load, due to the observation that giving enough buffer space, there is limited effect of changing the thresholds at loads below 90% of the link capacity. Furthermore, we chose two series where we had two substantially different min_{th} values to ensure that the observations made were independent of the min_{th} value. The results of these experiments are shown in Figures B.6, B.7, and B.8, and Table B.7 shows the summary statistics.

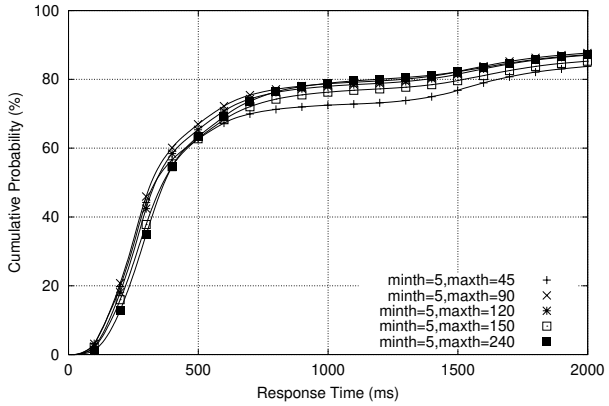


(a) min_{th}=5

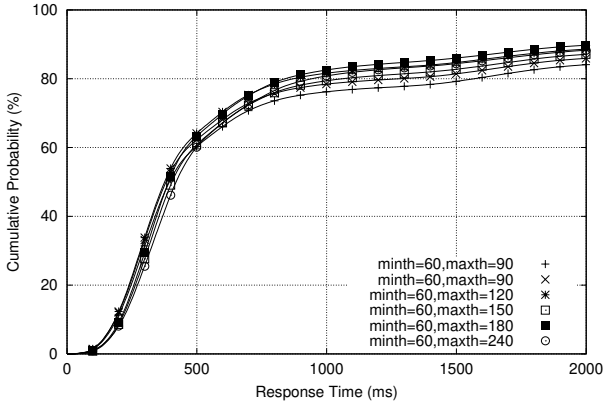


(b) min_{th} = 60

Figure B.6: *Plots with max_{th} at 90% load.*



(a) min_{th}=5



(b) $min_{th} = 60$

Figure B.7: *Plots with max_{th} at 98% load.*

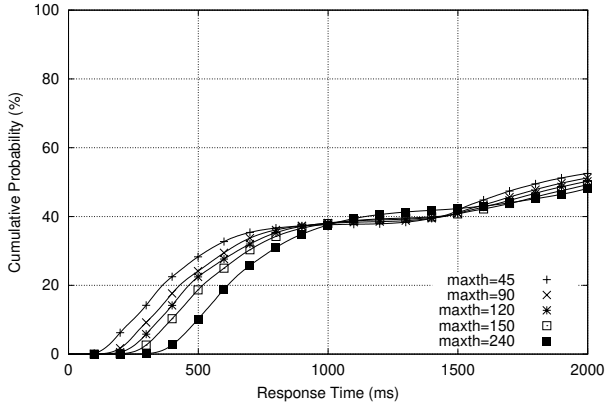


Figure B.8: *Plot with max_{th} at 110%.*

load (%)	min_{th}	max_{th}	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	5	45	1072	2.07	9.88	265	87.5	6.29	3.59	2.57
90	5	90	1087	1.59	12.8	267	89.2	5.69	3.09	2.03
90	5	120	1069	1.16	12.3	262	91.2	4.91	2.38	1.49
90	5	150	1086	1.43	16.1	270	89.8	5.34	2.89	1.94
90	5	180	1091	1.46	18.2	276	89.8	5.30	2.90	2.00
90	5	240	1085	1.09	18.3	272	91.3	4.81	2.37	1.55
90	60	90	1085	0.79	29.5	291	92.1	4.34	2.04	1.54
90	60	120	1084	0.38	28.9	285	94.0	3.74	1.26	0.96
90	60	150	1089	0.51	33.9	298	93.0	4.15	1.61	1.21
90	60	180	1068	0.38	30.9	289	94.0	3.81	1.26	0.96
90	60	240	1089	0.40	35.1	297	93.5	4.02	1.38	1.05
98	5	45	1157	5.84	18.7	334	72.5	11.3	8.03	8.18
98	5	90	1149	4.18	24.3	319	78.7	8.94	6.41	5.91
98	5	120	1170	4.20	30.8	335	78.0	9.05	6.84	6.11
98	5	150	1165	4.76	40.2	365	76.3	9.01	7.06	7.64
98	5	240	1175	3.74	49.5	370	78.9	8.29	6.56	6.24
98	60	90	1174	5.30	57.0	395	76.2	7.87	7.13	8.77
98	60	120	1165	3.31	58.9	376	81.6	7.00	5.42	5.95
98	60	150	1177	3.61	70.4	405	79.4	7.69	6.13	6.76
98	60	180	1172	2.63	68.8	392	82.5	7.19	5.09	5.16
98	60	240	1170	3.04	80.2	421	80.8	7.45	5.45	6.26
110	5	45	1180	21.6	36.5	1827	37.6	14.9	10.4	37.1
110	5	90	1188	19.7	75.0	1919	38.0	13.2	12.5	36.3
110	5	120	1188	19.2	100	1984	38.1	12.1	13.6	36.2
110	5	150	1188	18.3	126	2046	38.0	11.3	14.7	36.0
110	5	240	1188	16.6	195	2136	37.5	10.6	16.2	35.6

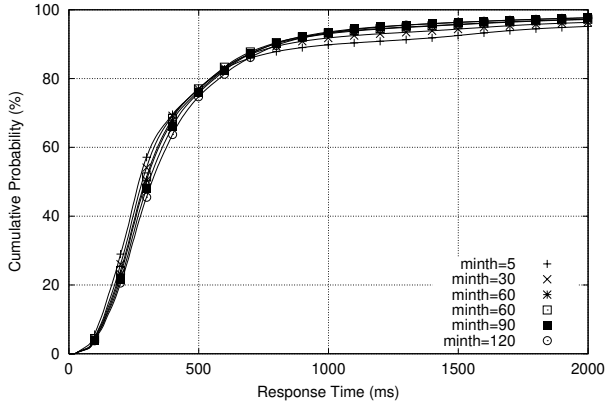
Table B.7: *RED experiments with* max_{th}*, where* $qlen = 480$*,* $max_p = 1/10$*, and* $w_q = 1/512$.

B.3.3 Experiments with min_{th}

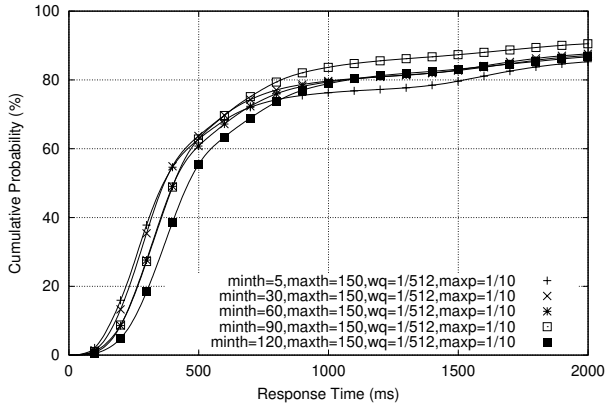
As with max_{th} we ran a series of experiments with min_{th}. The max_{th} was kept fixed at 150 packets, while min_{th} was changed from 5 to 120 packets. The results of these experiments are shown in Figure B.9 and the summary statistics are given in Table B.8.

load (%)	min_{th}	max_{th}	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	5	150	1086	1.43	16.1	270	89.8	5.34	2.89	1.94
90	30	150	1091	0.89	24.4	283	91.8	4.54	2.12	1.53
90	60	150	1089	0.51	33.9	298	93.0	4.15	1.61	1.21
90	60	150	1083	0.53	32.8	293	93.2	4.01	1.52	1.21
90	90	150	1090	0.33	40.9	308	93.5	4.13	1.28	1.05
90	120	150	1091	0.28	46.4	320	93.3	4.39	1.28	1.07
98	5	150	1165	4.76	40.2	365	76.3	9.01	7.06	7.64
98	30	150	1173	3.71	50.4	368	79.7	7.90	6.15	6.25
98	60	150	1177	3.61	70.4	405	79.4	7.69	6.13	6.76
98	90	150	1169	2.23	78.0	405	83.7	6.85	4.70	4.77
98	120	150	1181	3.63	101	461	78.9	7.78	5.70	7.58
110	5	150	1188	18.3	126	2046	38.0	11.3	14.7	36.0
110	30	150	1188	17.9	127	1982	39.1	11.2	14.5	35.2
110	60	150	1188	20.9	135	2367	35.1	10.4	14.7	39.8
110	60	150	1188	20.2	135	2308	35.8	10.6	14.7	38.8
110	90	150	1188	17.7	134	1932	40.4	10.7	14.2	34.7
110	120	150	1188	21.0	140	2233	37.6	9.90	13.7	38.8

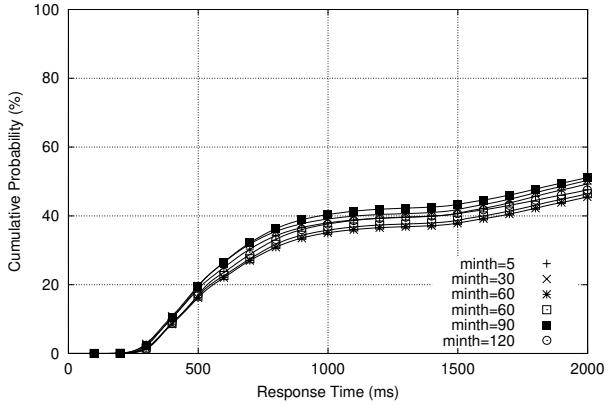
Table B.8: *Summary statistics for RED experiments with min_{th}, where qlen = 480, $max_p = 1/10$, and $w_q = 1/512$.*



(a) load = 90%



(b) load = 98%



(c) load=110%

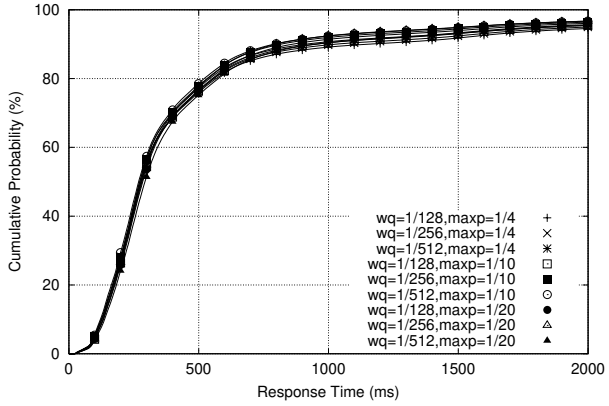
Figure B.9: *Plots with min_{th} experiments.*

B.3.4 Experiments with w_q and max_p

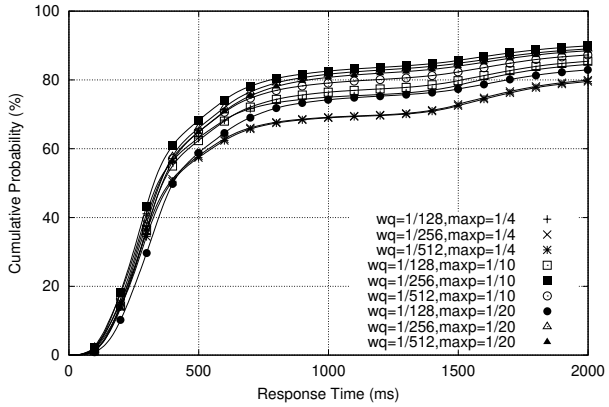
The parameters w_q and max_p were tested in a combined series of experiments, where the threshold values were kept fixed at $(30, 90)$. We tried a series of 3 different values of each of the two parameters. For w_q this means values ranging from 1/512 to 1/128 and for max_p this meant values from 1/20 to 1/4. Figure B.10 shows the response time CDF plots for the experiments and Table B.9 gives the summary statistics.

load (%)	$\frac{1}{w_q}$	$\frac{1}{max_p}$	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	128	4	1104	1.65	18.8	278	89.1	5.41	3.14	2.30
90	256	4	1088	1.25	17.7	273	90.7	4.93	2.60	1.74
90	512	4	1091	1.47	20.3	277	89.8	5.18	2.97	2.04
90	128	10	1111	1.26	22.3	282	90.4	5.02	2.67	1.87
90	256	10	1084	0.93	19.7	274	92.0	4.43	2.13	1.45
90	512	10	1079	0.83	20.2	269	92.4	4.30	1.98	1.32
90	128	20	1102	1.00	24.6	285	91.4	4.65	2.29	1.65
90	256	20	1099	0.78	23.2	280	92.4	4.28	1.95	1.34
90	512	20	1115	1.17	28.5	293	90.4	5.05	2.64	1.91
98	128	4	1161	5.72	29.8	343	75.0	9.54	7.18	8.29
98	256	4	1158	7.47	34.3	387	69.1	10.8	8.77	11.3
98	512	4	1165	7.28	36.8	394	69.0	10.6	9.07	11.4
98	128	10	1179	4.71	41.5	362	76.4	9.00	7.19	7.37
98	256	10	1163	3.09	34.8	328	82.6	7.33	5.39	4.70
98	512	10	1164	4.09	39.4	349	79.1	8.18	6.33	6.39
98	128	20	1179	5.82	56.1	401	74.2	8.69	7.40	9.65
98	256	20	1171	3.29	45.3	352	81.7	7.37	5.57	5.39
98	512	20	1173	3.51	48.0	360	80.7	7.75	5.77	5.73
110	128	4	1183	19.3	50.8	1771	38.2	16.0	11.4	34.4
110	256	4	1185	21.4	54.8	2121	33.4	15.4	12.0	39.2
110	512	4	1183	20.4	53.9	1954	35.6	15.1	12.1	37.3
110	128	10	1188	19.9	75.3	1897	38.9	12.7	12.2	36.2
110	256	10	1187	20.2	73.9	1871	39.2	12.7	12.1	35.9
110	512	10	1187	19.7	76.0	1852	39.4	12.9	12.2	35.6
110	128	20	1188	21.1	81.6	1780	42.9	10.0	11.0	36.1
110	256	20	1188	20.0	81.7	1648	44.7	10.6	10.7	34.0
110	512	20	1188	21.0	81.5	1736	43.4	10.2	10.7	35.7

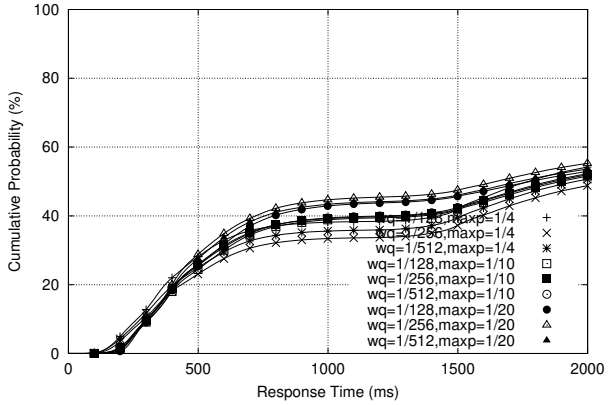
Table B.9: *Summary statistics of RED experiments with w_q and max_p. Other parameters are fixed as follows: qlen = 480, $min_{th} = 5$, and $max_{th} = 90$.*



(a) load = 90%



(b) load = 98%



(c) load = 110%

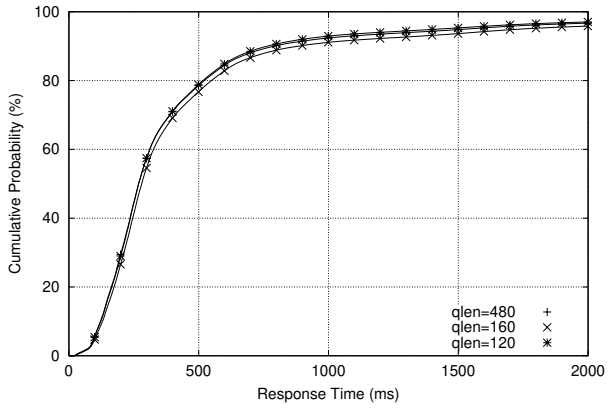
Figure B.10: *Experiments with w_q and max_p.*

B.3.5 Experiments with *qlen*

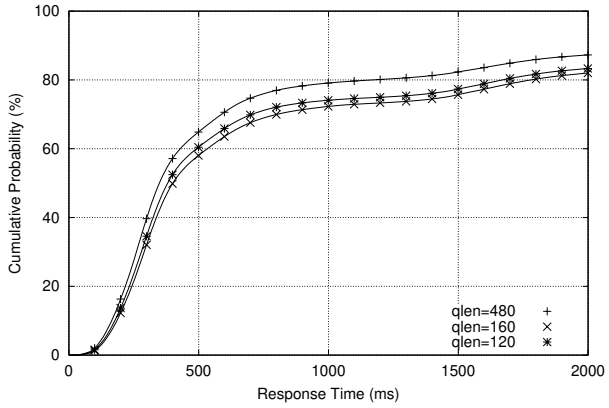
A final series of experiments evaluated the effect of reducing the queue length from the infinite 480 packets to something closer to the maximum threshold. As a basis for choosing the values of 120 and 160, we used the measurements of the queue length that were made during our previous experiments. The results are shown in Figure B.11 and Table B.10 gives the summary statistics.

load (%)	*qlen*	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	480	1079	0.83	20.2	269	92.4	4.30	1.98	1.32
90	160	1093	1.11	22.2	281	91.1	4.72	2.44	1.72
90	120	1066	0.72	18.8	269	92.9	4.11	1.75	1.21
98	480	1164	4.09	39.4	349	79.1	8.18	6.33	6.39
98	160	1175	5.92	46.3	401	72.3	9.70	8.20	9.78
98	120	1171	5.48	44.3	381	74.1	9.23	7.67	9.01
110	480	1187	19.7	76.0	1852	39.4	12.9	12.2	35.6
110	160	1188	19.5	76.6	1871	39.0	13.0	12.2	35.8
110	120	1188	19.4	77.4	1888	38.6	13.1	12.4	35.8

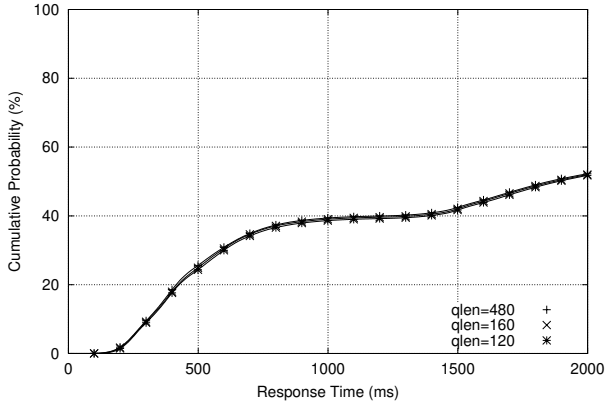
Table B.10: *Summary statistics for RED experiments with reduced queue sizes. Other parameters are fixed as follows:* $min_{th} = 30$, $max_{th} = 90$, $w_q = 1/512$, *and* $max_p = 1/10$.



(a) load = 90%



(b) load = 98%



(c) load = 110%

Figure B.11: *Experiments with different values of qlen at offered loads from 90-110%. Other parameters are fixed as follows:* $min_{th} = 30$, $max_{th} = 90$, $w_q = 1/512$, *and* $max_p = 1/10$.

B.3.6 "Good" and "Bad" Parameter Settings

Finally based on the series of RED experiments we are able to empirically determine "good" and "bad" RED parameter settings. Table B.11 lists the "good" parameter settings. We have made a subjective choice for the best overall response time at 90 and 98% load. Furthermore, we list parameter settings where we have optimized for highest linkutilization and lowest drop rate. Figure B.12 shows the response time CDF and Table B.12 the summary statistics for each of these parameter settings at 90-110% load. These plots also includes, as a reference point, the optimal response time measured on the unconstrained network.

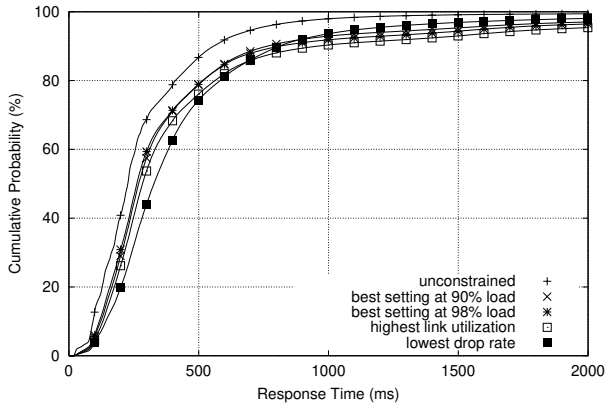
load (%)	min_{th}	max_{th}	w_q	max_p	*qlen*	Notes
90	30	90	1/512	1/10	120	best overall response (90%)
90	5	90	1/128	1/20	480	best overall response (98%)
90	30	90	1/512	1/20	480	highest link utilization
90	120	360	1/512	1/10	480	lowest drop rate
98	30	90	1/512	1/10	120	best overall response (90%)
98	5	90	1/128	1/20	480	best overall response (98%)
98	30	180	1/512	1/10	480	highest link utilization
98	90	150	1/512	1/10	480	lowest drop rate
110	30	90	1/512	1/10	120	best overall response (90%)
110	5	15	1/512	1/20	60	best overall response (110%)

Table B.11: *Empirically determined "good" RED parameter values.*

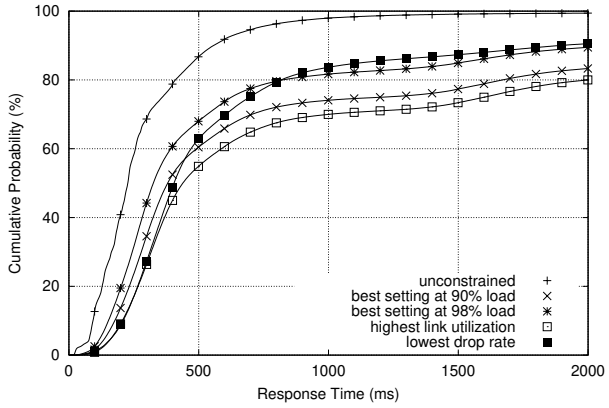
load (%)	**Notes**	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
-	unconstrained	994	0	0	227	98.0	1.44	0.23	0.35
90	best 90%	1066	0.72	18.8	269	92.9	4.11	1.75	1.21
90	best 98%	1061	1.03	12.9	262	91.9	4.59	2.17	1.37
90	highest link util	1102	1.23	23.3	284	90.4	5.03	2.73	1.88
90	lowest drop rate	1094	0.14	53.8	328	93.6	4.46	1.03	0.89
98	best 90%	1171	5.48	44.3	381	74.1	9.23	7.67	9.01
98	best 98%	1165	3.34	33.0	326	81.7	7.77	5.59	4.93
98	highest link util	1181	6.27	60.1	439	70.0	10.0	8.76	11.2
98	lowest drop rate	1169	2.23	78.0	405	83.7	6.85	4.70	4.77
110	best 90%	1188	19.4	77.4	1888	38.6	13.1	12.4	35.8
110	best 110%	1154	24.9	13.5	1849	39.3	12.5	9.25	38.9

Table B.12: *Summary statistics on "good" RED experiments. See Table B.11 for parameter settings.*

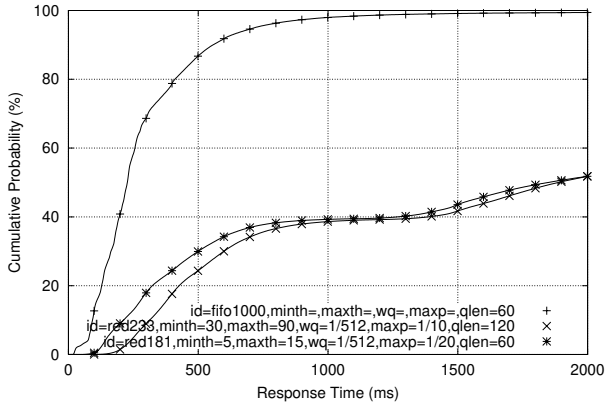
Secondly we found experiments that had plausible parameter settings that caused a significant reduction in response time performance. The experiment configurations for these are listed in Table B.14 and we refer to them as the "bad" parameter settings. As a reference for the performance impact of choosing a bad parameter setting we have included the "good" parameter setting for each of the offered loads



(a) Load = 90%



(b) Load = 98%



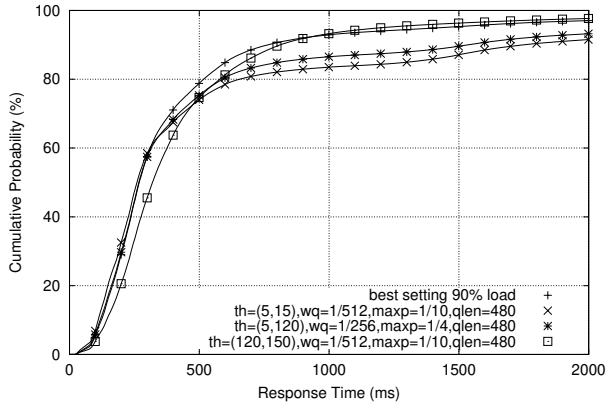
(c) Load = 110%

Figure B.12: *"Good" settings.*

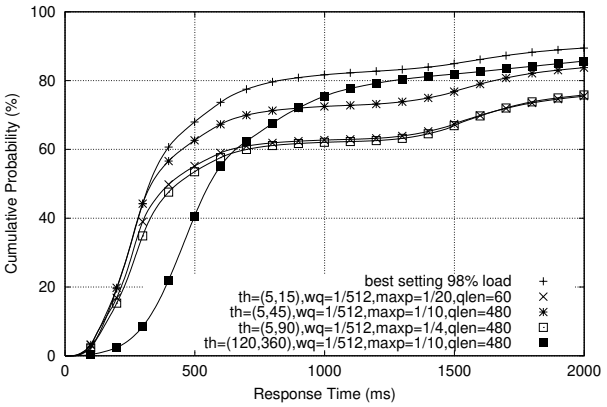
90% and 98%. Figure B.13 shows the response time CDFs and Table B.14 lists summary statistics.

load (%)	min_{th}	max_{th}	w_q	max_p	$qlen$	Notes
90	30	90	1/512	1/10	120	best overall response (90%)
90	5	15	1/512	1/10	480	small buffer size
90	5	120	1/256	1/4	480	high max_p value
90	120	150	1/512	1/10	480	large queue size
98	5	90	1/128	1/20	480	best overall response (98%)
98	5	15	1/512	1/20	60	default setting
98	5	45	1/512	1/10	480	small buffer size 2
98	5	90	1/512	1/4	480	high max_p value
98	120	360	1/512	1/10	480	large buffer size

Table B.13: *Empirically determined "bad" RED parameter values.*



(a) Load = 90%



(b) Load = 98%

Figure B.13: *"Bad" settings.*

load (%)	**Notes**	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
90	best 90%	1066	0.72	18.8	269	92.9	4.11	1.75	1.21
90	small buffer	1068	3.15	7.11	260	83.5	8.02	4.79	3.66
90	high max_p value	1082	2.30	10.6	268	86.5	6.67	3.86	2.92
90	large buffer	1091	0.28	46.4	320	93.3	4.39	1.28	1.07
98	best 98%	1165	3.34	33.0	326	81.7	7.77	5.59	4.93
98	default setting	1142	11.2	12.2	402	62.7	12.8	9.56	14.9
98	small buffer	1157	5.84	18.7	334	72.5	11.3	8.03	8.18
98	high max_p value	1165	9.22	22.5	434	62.1	13.8	9.78	14.4
98	large buffer	1182	3.06	147	557	75.5	10.2	6.05	8.28

Table B.14: *Summary statistics on empirically determined "bad" RED experiments. See Table B.13.*

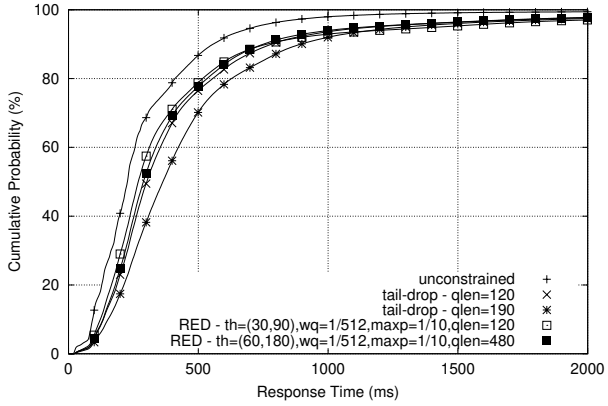
B.4 Comparing RED and Tail-drop

This final section presents the RED versus tail-drop comparison. Based on the empirical evaluation we have subjectively selected a RED and a tail-drop configuration that we find to give the best overall performance independent of offered load level. In Figure B.14 we show the best overall configurations combined with tail-drop configured with a queue length of 1-2 times the bandwidth-delay product. Furthermore, for each offered load we show the best RED parameter setting for that load level. And finally as a performance reference we have added the response time CDF measured on a network with the unconstrained link.

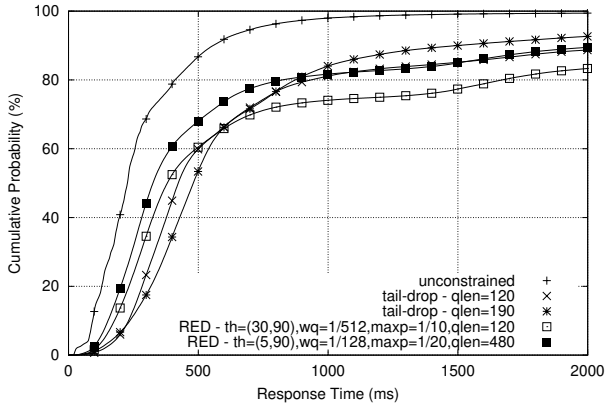
Table B.15 lists the configuration of each of the experiments shown in the Figures and Tables B.16- B.18 gives the summary statistics for the experiments shown in the plots.

Note	**Parameters**
best tail-drop overall	$qlen = 120$
best tail-drop 1-2·bwdp	$qlen = 190$
best RED overall	$th = (30, 90), max_p = 1/10, w_q = 1/512, qlen = 120$
best RED at 90%	$th = (60, 180), max_p = 1/10, w_q = 1/512, qlen = 480$
best RED at 98%	$th = (5, 90), max_p = 1/20, w_q = 1/128, qlen = 480$
best RED at 110%	$th = (30, 90), max_p = 1/20, w_q = 1/256, qlen = 480$

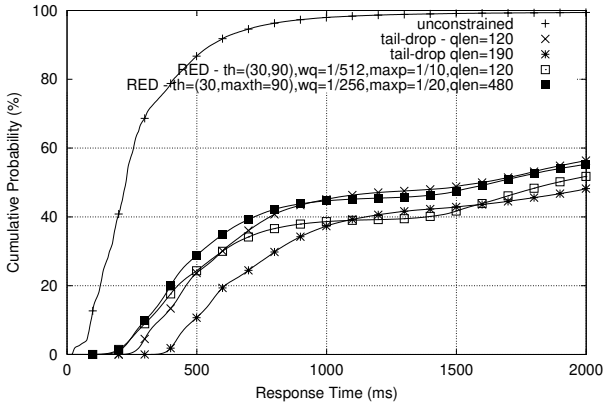
Table B.15: *RED vs tail-drop.*



(a) 90% load



(b) 98% load



(c) 110% load

Figure B.14: *Tail-Drop and RED comparisons at different offered load levels.*

Note	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
uncongested	994	0	0	227	98.0	1.44	0.23	0.35
best tail-drop overall	1085	0.34	40.3	302	93.6	4.05	1.28	1.04
tail-drop w/ 1-2·bwdp	1106	0.21	66.6	364	92.0	5.45	1.32	1.21
best tail-drop overall	1066	0.72	18.8	269	92.9	4.11	1.75	1.21
best RED at 90%	1068	0.38	30.9	289	94.0	3.81	1.26	0.96

Table B.16: *RED compared with tail-drop at 90% offered load.*

Note	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
uncongested	994	0	0	227	98.0	1.44	0.23	0.35
best tail-drop overall	1178	3.07	84.8	423	81.1	7.58	5.16	6.13
tail-drop w/ 1-2·bwdp	1166	1.28	119	481	84.0	8.61	3.34	3.99
best RED overall	1171	5.48	44.3	381	74.1	9.23	7.67	9.01
best RED 98%	1165	3.34	33.0	326	81.7	7.77	5.59	4.93

Table B.17: *RED compared with tail-drop at 98% offred load.*

Note	$\frac{KB}{s}$	% drops	μ *qlen*	η rsp-time (ms)	i_1	i_2	i_3	i_4
uncongested	994	0	0	227	98.0	1.44	0.23	0.35
best tail-drop overall	1188	17.0	112	1601	45.1	11.3	13.0	30.6
tail-drop w/ 1-2·bwpd	1188	19.3	183	2153	37.3	10.9	14.7	37.0
best RED overall	1188	19.4	77.4	1888	38.6	13.1	12.4	35.8
best RED at 110%	1188	20.0	81.7	1648	44.7	10.6	10.7	34.0

Table B.18: *RED compared with tail-drop at 110% offered load.*

Appendix C

Alternate Queueing

This note describe a small difference between the RED implementation used is in Alternate Queueing (ALTQ) [36] and the RED algorithm described in [29]. Furthermore we describe how we fit the parameters to avoid any impact of this difference.

In general the implementation follows the algorithm that we described in Section 2.1 except for on small difference in the calculation of p_a. The original definition of p_a is given by:

$$p_a = p_b/(1 - count \cdot p_b) \tag{C.1}$$

The ALTQ implementation differs slightly by using the following definition:

$$p'_a = p_b/(2 - count \cdot p_b) \tag{C.2}$$

The question is how does this effect the RED algorithm and its parameters. We have determined that we can avoid any impact of this difference by fitting the parameter max_p. First we observe that p_b should be doubled for the implementation of p_a to be equivalent with the definition:

$$p_a = \frac{p_b}{1 - count \cdot p_b} = \frac{2p_b}{2 - count \cdot 2p_b} \tag{C.3}$$

To get $2p_b$ by adjusting the parameters to RED, then we need double the value of max_p:

$$2p_b = \frac{2max_p(avg - min_{th})}{max_{th} - min_{th}} \tag{C.4}$$

Whenever describing the value of max_p in this thesis we always refer to the value in agreement with the original definition in [29].

Bibliography

[1] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proceeding of ACM SIGCOMM*, pages 263–274, Cambridge, MA, USA, September 1999.

[2] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP congestion control. RFC, Internet Engineering Task Force, April 1999. Status: STANDARD.

[3] F. M. Anjum and L. Tassiulas. Balanced RED: An algorithm to achieve fairness in the internet. Technical report, University of Maryland at College Park, March 1999. A short version of the paper appeared at INFOCOMM 99.

[4] G. Banga. Measuring the capacity of a web server. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 61–71, Monterey, CA, USA, December 1997.

[5] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS*, pages 151–160, Madison, WI, USA, June 1998.

[6] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, 1994.

[7] T. Berners-Lee and D. W. Connolly. RFC 1866: Hypertext Markup Language — 2.0. RFC, Internet Engineering Task Force, November 1995. Status: PROPOSED STANDARD.

[8] T. Berners-Lee, R. Fielding, and H. F. Nielsen. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0. RFC, Internet Engineering Task Force, May 1996. Status: INFORMATIONAL.

[9] 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, and L. Zhang. RFC 2309: Recommendations on queue management and congestion avoidance in the Internet. RFC, Internet Engineering Task Force, April 1998. Status: INFORMATIONAL.

[10] R. Cáceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area TCP/IP conversations. In *Proceedings of ACM SIGCOMM*, pages 101–112, Zurich, Switzerland, 1991.

[11] Cisco Systems. `http://www.cisco.com`.

[12] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of ACM SIGMETRICS*, pages 160–169, Philadelphia, PA, USA, May 1996.

[13] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[14] P. B. Danzig and S. Jamin. tcplib: A library of TCP internetwork traffic characteristics. Technical report, Computer Science Department, University of Southern California, Los Angeles, California 90089-0781, 1991.

[15] P. B. Danzig, S. Jamin, R. Cáceres, D. J. Mitzel, and D. Estrin. An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking: Research and Experience*, 3(1):1–26, March 1992.

[16] A. Demers, S. Keshav, and S. Shenker. Analysis and simulations of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, Austin, TX, USA, 1989.

[17] `http://adm.ebone.net/~smd/red-1.html`.

[18] A. Feldmann. Characteristics of TCP connection arrivals. Technical report, AT&T Labs-Research, December 1998.

[19] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of ip traffic: a study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM*, pages 301–313, Cambridge, MA, USA, 1999.

[20] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. BLUE: A new class of active queue management algorithms. Technical Report 99/387, University of Michigan, 1999.

[21] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring RED gateway. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1320–1328, March 1999.

[22] R. Fielding, J. Gettys, J. Mogul, H. F. Nielsen, and T. Berners-Lee. RFC 2068: Hypertext Transfer Protocol — HTTP/1.1. RFC, Internet Engineering Task Force, January 1997. Status: PROPOSED STANDARD.

[23] V. Firoiu and M. Bordon. A study of active queue management for congestion control. In *Proceedings of IEEE INFOCOM*, pages 1435–1444, Tel Aviv, Israel, March 2000.

[24] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, October 1994. This issue of CCR incorrectly has "1995" on the cover instead of "1994".

[25] S. Floyd. RED: Discussions of setting parameters. `http://www.aciri.org/floyd/REDparameters.txt`, 1997.

[26] S. Floyd and K. Fall. NS simulator tests for random early detection (RED) queue management. Technical report, Lawrence Berkeley Laboratory, One Cyclotron Road, Berkeley, CA 94704, Appril 1997.

[27] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 7(4), August 1999.

[28] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3(2):115–156, September 1992.

[29] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[30] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, 2(2):122–136, April 1994.

[31] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):342–356, August 1995.

[32] Internet Research Task Force (IRTF). `ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail`, 1998.

[33] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP extensions for high performance. RFC, Internet Engineering Task Force, May 1992. Status: PROPOSED STANDARD.

[34] V. Jacobson and M. Karels. Congestion avoidance and control. In *Procceedings of ACM SIGCOMM*, pages 314–329, Stanford, CA, USA, 1988.

[35] R. Jain and S. Routhier. Packet trains-measurements and a new model for computer network traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6), September 1986.

[36] C. Kenjiro. A framework for alternate queueing: Towards traffic management by pc-unix based routers. In *Proceedings of USENIX Annual Technical Conference*, pages 247–258, New Orleans, LA, USA, June 1998.

[37] S. Keshav. REAL: A network simulator. Technical Report 88/472, Department of Electrical Engineering and Computer Science, Computer Science Department, University of California at Berkeley, Berkeley, California 1988, 1988.

[38] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic(extended version). *IEEE/ACM Transactions on networking*, 2(1), February 1994.

[39] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM*, pages 127–137, Cannes, France, September 1997.

[40] B. A. Mah. An empirical model of HTTP network traffic. In *Proceedings of IEEE INFOCOM*, pages 592–600, Kobe, Japan, April 1997.

[41] B. B. Mandelbrot. Long-run linearity, locally gaussian processes, h-spectra and infinite variances. *International Economic Review*, 10:82–113, 1969.

[42] A. Mankin. Random drop congestion control. In *Proceedings of ACM SIGCOMM*, pages 1–7, Philadelphia, PA, USA, September 1990.

[43] A. Mankin and K. Ramakrishnan. RFC 1254: Gateway congestion control survey. RFC, Internet Engineering Task Force, July 1991. status: INFORMATIONAL.

[44] M. May, J. Balot, C. Diot, and B. Lyles. Reasons not do deploy RED. In *Proceedings of 7th. International Workshop on Quality of Service (IWQoS'99)*, pages 260–262, London, UK, June 1999.

[45] M. May, T. Bonald, and J. Bolot. Analytic evaluation of RED performance. In *Proceedings of IEEE INFOCOM*, pages 1415–1424, Tel-Aviv, Israel, March 2000.

[46] S. McCanne and S. Floyd. ns network simulator. `http://www.isi.edu/nsnam/ns`.

[47] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese. Application performance pitfalls and TCP's Nagle algorithm. In *In Workshop on Internet Server Performance*, May 1999.

[48] J. C. Mogul. The case for persistent-connection http. In *Proceedings of the ACM SIGCOMM*, pages 299–313, Boston, MA, USA, August/September 1995.

[49] J. Nagle. RFC 896: Congestion control in IP/TCP internetworks. RFC, Internet Engineering Task Force, January 1984.

[50] A. Neidhardt. Traffic source models for the bestavros and crovella data. *Private Communication*, 1996.

[51] Netstat. `http://www.netstat.net` (service no longer available).

[52] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of ACM SIGCOMM*, Cannes, France, 1997.

[53] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOMM 99*, pages 1346–1355, San Francisco, CA, USA, March 1999.

[54] M. Parris, K. Jeffay, and F. D. Smith. Lightweight active router-queue management for multimedia networking. In *Proceedings of Multimedia Computing and Networking*, SPIE Proceedings Series, pages 162–174, San Jose, CA, USA, January 1999.

[55] V. Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[56] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[57] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):225–244, June 1995.

[58] J. Postel. RFC 791: Internet Protocol. RFC, Internet Engineering Task Force, September 1981. Status: STANDARD.

[59] B. Raynolds. `http://null0.qual.net/brad/papers/reddraft.htm`. link now broken.

[60] L. Rizzo. Dummynet. `http://www.iet.unipi.it/~luigi/ip_dummynet`.

[61] S. Shenker, L. Zhang, and D. D. Clark. Some observations on the dynamics of a congestion control algorithm. *Computer Communications Review*, 20(5):30–39, October 1990.

[62] F. D. Smith, F. H. Compos, K. Jeffay, and D. Ott. What TCP/IP protocol headers can tell us about the web. In *Proceedings of ACM SIGMETRICS*, pages 245–256, Cambridge, MA, USA, June 2001.

[63] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 1994.

[64] M. S. Taqqu and J. Levy. *Dependence in Probability and Statistics: A Survey of Recent Results*, pages 73–90. Birkhäuser, 1986.

[65] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area internet traffic patterns and characteristics. *IEEE Network*, 11(6), November/December 1997.

[66] C. Villamizar and C. Song. High performance TCP in ANSNET. *ACM Computer Communications Review*, 24(5):45–60, October 1994.

[67] Z. Wang and P. Cao. Persistent connection behavior of popular browsers. `http://www.cs.wisc.edu/~cao/papers/persistent-connection.htm`, December 1998.

[68] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. In *Proceedings of ACM SIGCOMM*, pages 100–113, Cambridge, MA, USA, August 1995.

[69] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[70] L. Zhang and D. D. Clark. Oscillating behavior of network traffic: A case study simulation. *Internetworking: Research and Experience*, 1:101–112, 1990.

[71] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of ACM SIGCOMM*, pages 133–147, Zurich, Switzerland, September 1991.

[72] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Hafner Publishing Company, New York, NY, USA, 1949.