Traffic Sensitive Active Queue Management for Improved Quality of Service

by

Vishal Phirke

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2002

APPROVED:

Professor Mark Claypool, Major Thesis Advisor

Professor Robert Kinicki, Major Thesis Advisor

Professor David Finkel, Thesis Reader

Professor Micha Hofri, Head of Department

Abstract

The Internet, traditionally FTP, e-mail and Web traffic, is increasingly supporting emerging applications such as IP telephony, video conferencing and online games. These new genres of applications have different requirements in terms of throughput and delay than traditional applications. For example, interactive multimedia applications, unlike traditional applications, have more stringent delay constraints and less stringent loss constraints. Unfortunately, the current Internet offers a monolithic best-effort service to all applications without considering their specific requirements. Adaptive RED (ARED) is an Active Queue Management (AQM) technique, which optimizes the router for throughput. Throughput optimization provides acceptable QoS for traditional throughput sensitive applications, but is unfair for these new delay sensitive applications. While previous work has used different classes of QoS at the router to accommodate applications with varying requirements, thus far all have provided just 2 or 3 classes of service for applications to choose from. We propose two AQM mechanisms to optimize router for better overall QoS. Our first mechanism, RED-Worcester, is a simple extension to ARED in order to tune ARED for better average QoS support. Our second mechanism, RED-Boston, further extends RED-Worcester to improve the QoS for all flows. Unlike earlier approaches, we do not predefine classes of service, but instead provide a continuum from which applications can choose. We evaluate our approach using NS-2 and present results showing the amount of improvement in QoS achieved by our mechanisms over ARED.

Acknowledgements

I thank Dr. Mark Claypool and Dr. Robert Kinicki, my advisors, for their continual support and encouragement throughout this work. They always gave full consideration to my ideas and guided me whenever I appeared lost in doubts. Without them, this thesis would not have been possible.

I thank Dr. David Finkel for being the reader for my thesis and providing valuable comments to improve this work. I would also like to extend special thanks to my friends, Aditya, Anuj, Dipen and Mandar, for helping me in their own ways and making life easier for me. I thank all my friends at WPI who have made my stay here a memorable thing in my life.

I dedicate this work to my parents, whose unconditional love and moral support has helped me throughout my life.

Contents

1 Introduction **1**

2 Related Work **7**

2.1 Simple AQM Approaches 8

2.2 Throughput Fairness Approaches 10

2.3 Class Based Approaches 11

2.4 Integrated Services 13

2.5 Queue Law 14

3 RED-Worcester **16**

3.1 Delay Hints 17

3.2 Mechanism 18

3.2.1 Moving Target 18

3.3 Overhead 18

3.4 Evaluation 19

3.4.1 Simulation Setup 19

3.4.2 Analysis 22

4 RED-Boston **29**

4.1 Delay Hints 30

4.2 Mechanism 32
4.2.1 Delay Hint Based Drop Probability 32
4.2.2 Weighted Insert 33
4.2.3 Algorithm 33
4.3 Overhead 34
4.4 Evaluation 34
4.4.1 Simulation Set - 1 36
4.4.2 Simulation Set - 2 41

5 Conclusions **46**

6 Future Work **48**

List of Figures

1.1 QoS Spectrum of Applications 2

1.2 Approaches for QoS in Internet Routers 3

2.1 Adaptive RED 9

2.2 Adaptive RED Algorithm for Adjusting Average Queue to the Target Queue 9

2.3 Queue Law 15

3.1 RED-Worcester 16

3.2 Network Topology 20

3.3 Simulation RED-Worcester and ARED: Traffic Mix 22

3.4 RED-Worcester's Moving Target 23

3.5 max_p Analysis I 24

3.6 max_p Analysis II 24

3.7 Average queue size in RED-Worcester 25

3.8 Percent packet drops 26

3.9 Normalized QoS for Throughput-Sensitive Flows 27

3.10 Normalized QoS for Delay-Sensitive Flows 28

4.1 Possible Hint Strategy for an Interactive Multimedia Application 31

4.2 RED-Boston Algorithm 35

4.3 Simulation Set-1: Traffic Mix 36

4.4 Simulation Set-1: Average Queue Size 37
4.5 Simulation Set-1: Queuing Delays 38
4.6 Simulation Set-1: Average Per Flow Percent of Packets Dropped 39
4.7 Simulation Set-1: Average Per Flow Throughput 39
4.8 Simulation Set-1: Normalized QoS for Throughput-Sensitive Flows . . . 40
4.9 Simulation Set-1: Normalized QoS for Delay Sensitive Flows 41
4.10 Simulation Set-2: Traffic Mix 42
4.11 Simulation Set-2: Queuing Delays 42
4.12 Simulation Set-2: Average Per Flow Percentage of Packets Dropped . . . 43
4.13 Simulation Set-2: Average Per Flow Throughput 43
4.14 Simulation Set-2: Normalized QoS 44

List of Tables

4.1 Simulation Set-2: QoS Parameters 44

Chapter 1

Introduction

The Internet today carries traffic for applications with a wide range of delay and throughput requirements as depicted in Figure 1.1. Traditional applications such as FTP and E-mail are primarily concerned with throughput and hence, can withstand large queues in Internet routers, which improves throughput. On the other hand, emerging applications such as IP telephony, video conferencing and networked games have different requirements in terms of throughput and delay than these traditional applications. In particular, interactive multimedia applications, unlike traditional applications, have more stringent delay constraints than loss constraints. Moreover, with the use of repair techniques [BFPT99, LC00, PCM00, PW99] packet losses can be partially or fully concealed, enabling multimedia applications to operate over a wide range of losses, and leaving end-to-end delays as the major impediment to acceptable quality. Thus, interactive multimedia applications prefer smaller queues in Internet routers. Web traffic is moderately sensitive to delay as well as throughput and hence, it falls in the middle of the spectrum and prefers medium queues in Internet routers.

Unfortunately, current Internet routers are not able to provide a choice in Quality of

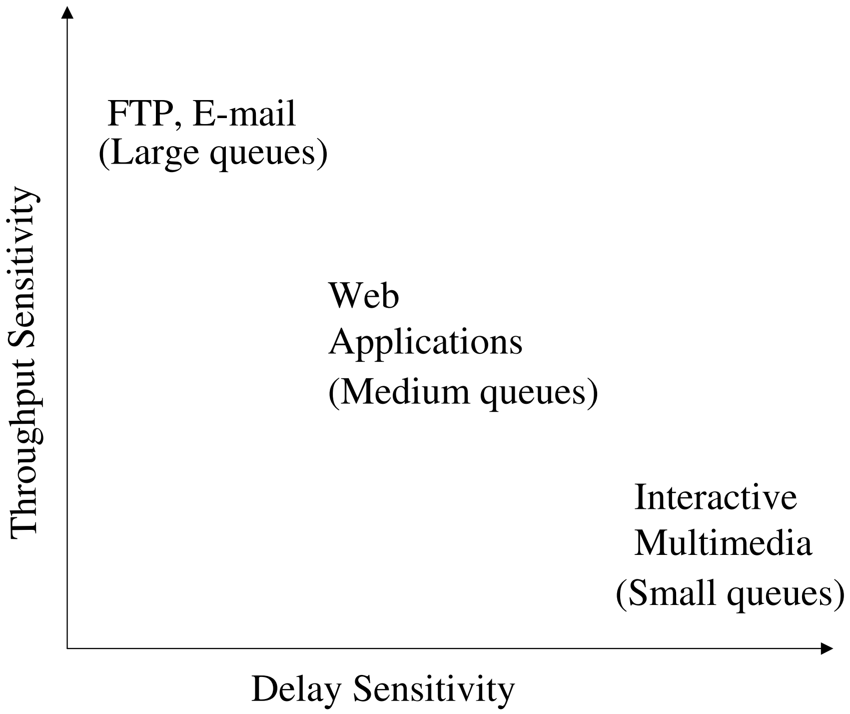


Figure 1.1: QoS Spectrum of Applications

Service (QoS).[1] Most of the current Active Queue Management (AQM) techniques focus on providing higher throughput at the router without much consideration for queuing delays. Current and proposed router queue mechanisms can be classified as in Figure 1.2 on the basis of the level of QoS support provided and the complexity of the router implementation.

Due to the simplicity of the FIFO queuing mechanism, drop-tail queues are the most widely used queuing mechanism in Internet routers today. When drop-tail buffers overflow, newly arriving packets are dropped regardless of the application constraints for the packet. To accommodate bursty traffic, drop-tail routers on the Internet backbone are over-provisioned with large FIFO buffers. When faced with persistent congestion, these drop-tail routers yield high delays for all flows passing through the bottlenecked router.

[1] Throughout this work, we use the term QoS to refer explicitly to *delay* and *throughput* provided by the network.

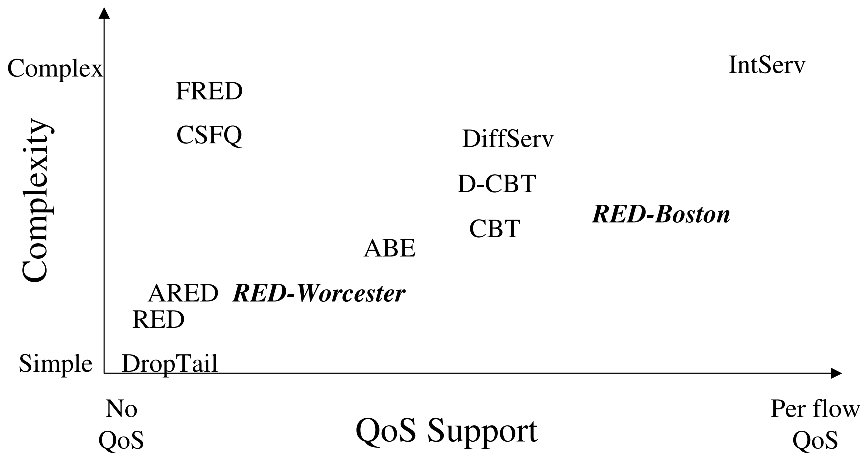


Figure 1.2: Approaches for QoS in Internet Routers

This best-effort service provides no consideration for interactive multimedia flows or even Web flows that can be severely affected by high delays. Clearly, drop-tail provides very limited, if any, QoS support.

RED [FJ93], probably the best-known AQM mechanism, attempts to keep the average queue size at the router low, while keeping throughput high. By detecting the onset of congestion earlier than drop-tail, RED avoids the global synchronization of TCP flows that hampers aggregate throughput. Adaptive RED (ARED) [FGS01] adjusts the RED operating parameters to accommodate a wider range of traffic loads. However, both RED and ARED provide equal treatment to incoming traffic and tend to be tuned for high throughput without any consideration for the traversing application's delay requirements.

Streaming multimedia applications avoid using TCP because TCP can suffer from bursty data rates and because TCP retransmits data lost due to packet drops at the router. This results in poorer quality multimedia due to increased delay and jitter. Instead, streaming media applications often choose UDP as their transport protocol [WCZ01]. However, UDP is unresponsive to packet drops that indicate congestion at the router. Flow Random Early Drop (FRED) [LM97] and Core-Stateless Fair Queuing (CSFQ) [SSZ98] routers are specifically designed to limit the bandwidth of unresponsive flows to achieve throughput fairness among flows. While these strategies provide good QoS for flows highly concerned about throughput, FRED and CSFQ ignore the willingness of interactive multimedia applications to accept reduced throughput if accompanied by reduced delay.

Since unresponsive UDP flows can lead to network congestion collapse and because UDP does not adjust its data rates when faced with network congestion, there is a strong movement to make streaming multimedia traffic *TCP-friendly* [FF99]. In [FF99], a TCP-friendly flow is defined as the flow whose arrival rate does not exceed the arrival rate of a conformant TCP connection in the same circumstances. In the near future all multimedia applications are likely to use TCP-friendly protocols that adjust their tranmission rates smoothly, such as in [FHPW00, RHE99, TZ99] without actually using TCP. Hence, this research assumes all flows, including multimedia flows, either use TCP or are TCP-friendly flows that respond to packet drops as indicators of congestion.

ABE [HKBT01] provides a queue management mechanism for low delay traffic. In ABE, delay-sensitive applications can sacrifice throughput for lower delays. However, ABE traffic classification is rigid in that applications are either delay-sensitive or throughput-sensitive without the applications themselves being able to choose relative degrees of sensitivity to throughput and delay.

CBT [PJS99] provides class-based treatment with guarantees on bandwidth limits for different classes. However, these classes are pre-determined and fixed, while practically, even within the multimedia traffic class, interactive multimedia applications have more stringent delay requirements than broadcast multimedia applications. DCBT with ChIPS [CC00] extends CBT by providing dynamic thresholds and lower jitter for multimedia traffic, but still limits all multimedia traffic to the same QoS.

DiffServ approaches, such as Assured Forwarding (AF) [HBWW99] and Expedited Forwarding (EF) [JNP99], try to give differentiated service to traffic aggregates. However, they require complicated mechanisms to negotiate service level agreements. In addition, DiffServ architectures require traffic monitors, markers, traffic shapers, classifiers and droppers and framework to enable components to work together.

IntServ provides the best possible per flow QoS guarantees. However, it requires complex signaling and reservations via RSVP [Wro97] by all routers along a connection on a per-flow basis, making scalability difficult for global deployment.

We present two new AQM techniques called, *RED-Worcester*[2] and *RED-Boston*[2], to improve QoS support at the router. In our approach, applications mark each packet with a suggested delay, referred to as a *delay hint*, indicating the relative importance of delay versus throughput. Our first AQM mechanism, *RED-Worcester*, is a simple extension to ARED and tries to meet the average performance requirements of incoming packets in terms of throughput and delay, thus improving the overall QoS support at the router. Our second AQM mechanism, *RED-Boston*, tries to meet the individual performance requirements of incoming packets, allowing them to choose lower throughput for lower delays or higher delays for higher throughput. *RED-Boston* uses *RED-Worcester* to adjust the average queue size as per traffic requirements and uses *weighted insert* and *delay hint based drop probability* mechanisms to provide individual delay-throughput trade-off to incoming packets. The service is still best-effort in that it requires no additional policing mechanisms, charging mechanisms or usage control. Under the proposed service, *RED-Worcester* and *RED-Boston* routers operate equally well under scenarios with only traditional traffic and operate better under scenarios with mixed or multimedia only traffic.

We evaluated *RED-Worcester* and *RED-Boston* via simulation under a variety of traffic mixes and compared their performance with ARED. The results show that as traffic mix changes from mostly throughput-sensitive to mostly delay-sensitive, *RED-Worcester* improves the average QoS support at the router whereas *RED-Boston* improves the QoS support for delay-sensitive as well as through-sensitive flows. We also evaluated *RED-*

[2]Similar to various versions of TCP, which are named after cities in Nevada, we have named our versions of the RED active queue management technique after cities in Massachusetts.

Boston with a traffic mix consisting of applications with a range of delay and throughput requirements and found that RED-Boston achieves a performance that is more suitable to all applications than performance under ARED.

The remainder of this thesis is organized as follows: Chapter 2 describes the related work; Chapter 3 presents implementation details of *RED-Worcester* and its evaluation; Chapter 4 presents implementation details of *RED-Boston* and its evaluation; Chapter 5 summarizes our work with conclusions and Chapter 6 discusses the possible future work.

Chapter 2

Related Work

Queues are used in routers to absorb transient bursts in incoming packet rates, allowing the router sufficient time for packet transmission. When the incoming packet rate is consistently higher than the router's outgoing packet rate, the queue size will increase, eventually exceeding available buffer space. When the buffer is full, some packets must be dropped. A simple solution is to drop the packets that are just arriving at the input port, i.e., if a packet arrives and finds the queue full it is dropped. This policy is known as drop-tail or tail-drop. Although drop-tail is simple to implement and has been in use in the Internet for many years, it can lead to poor performance and can cause global synchronization, lockouts, and full queues [Has89, FJ92]. Also, drop-tail gives uniform treatment to all flows, without considering the individual needs of the applications supported by these flows.

Different approaches have been proposed to address the performance problems of drop-tail. Section 2.1 discusses simple active queue management approaches such as RED and ARED. Section 2.2, explains active queue management approaches which try to distribute router's bandwidth fairly amongst all the competing flows. Section 2.3 covers class based approaches, which provides class based services to groups of flows. Section 2.4 discusses the IntServ approach, which provides per flow QoS guarantees by doing per flow reservations in the routers. Section 2.5 explains research on a queue law describing the relationship between average queue size and packet drop rates at the router.

2.1 Simple AQM Approaches

RED (Random Early Detection) [FJ93] is a proactive AQM technique that detects incipient congestion and provides feedback to end hosts by dropping packets early. The motivation behind RED is to keep the average queue size small, increase throughput, reduce bursty loss and global synchronization. RED operates based on an average queue length that is calculated using an exponential weighted average of the instantaneous queue length. RED drops packets with a probability depending on the average length of the queue. The drop probability increases from 0 to a maximum drop probability max_p as the average queue size increases from a minimum threshold min_{th} to the maximum threshold max_{th}. If the average queue size goes above the max_{th}, all packets are dropped.

In RED, the average queue size varies with the level of congestion and with the parameter settings [MBDL99, OLW99]. [CJOS00] shows that RED provides no advantages over traditional drop tail queuing for Web-only traffic. Using response time as a performance metric, [CJOS00] showed that below 90% load, there is no significant improvement in response times for Web traffic with RED over drop-tail.

Adaptive RED [FGS01] tries to perform better than RED over wider range of loads. Adaptive RED provides a predictable average delay by restricting the average queue size within a fixed target range. If the current average queue size goes above the target range, packets are dropped more aggressively from the queue to bring the average down. On the other hand, if the current average is below the target range, packets are dropped less aggressively to increase the average queue.

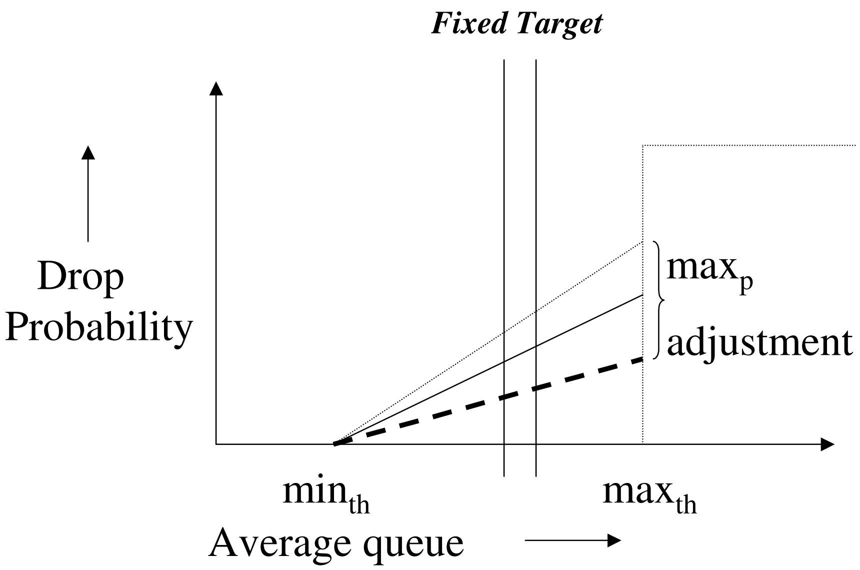


Figure 2.1: Adaptive RED

Every $interval$ **seconds:** *// 500ms, by default*
    if (q_{avg} > target and $max_p < 0.5$) then
        increase max_p:
            max_p += α *// about 0.1, by default*
    elseif (q_{avg} < target and $max_p > 0.01$) then
        decrease max_p:
            max_p $\times$= β *// 0.9, by default*

Figure 2.2: Adaptive RED Algorithm for Adjusting Average Queue to the Target Queue

Thus, ARED keeps RED's basic structure but adjusts max_p to keep the q_{ave} within a target range between min_{th} and max_{th} as shown in Figure 2.1. The default target is centered halfway between min_{th} and max_{th} with a range of 0.1 on either side. The ARED algorithm adjusts its target as depicted in Figure 2.2. In ARED, queue weight w_q used for calculating the average queue size is automatically set as a function of outgoing link's bandwidth. The maximum threshold max_{th} is set to three times min_{th}. Thus, the only parameter to choose is min_{th}. [FGS01] reports that min_{th} can be selected based on average queuing delay requirements at the router. However, pre-configured average queuing delays will then be fixed for all types of traffic mix even if they have varying

QoS requirements.

The first mechanism proposed in this thesis, RED-Worcester, dynamically configures the average queue size at the router to meet the average queuing delay requirements of the incoming packets. The second mechanism, RED-Boston, tries to provide individual delay-throughput tradeoff to the incoming packets, thus providing better QoS to all applications.

2.2 Throughput Fairness Approaches

Throughput fairness approches keep per flow states at the router in order to give a fair share of the throughput to all flows. Flow Random Early Drop (FRED) [LM97] and Core-Stateless Fair Queuing (CSFQ) [SSZ98] are two such approaches.

FRED is an extension to RED which tries to improve the throughput fairness between heterogeneous flows. It maintains an estimate of the average per flow buffer count, and favors the flows with fewer packets queued than the estimated average over the flows having more packets queued than the average per flow estimate. FRED uses per flow state variables, $qlen_i$ indicating number of packets buffered for the flow and $strike_i$ indicating the number of times a particular flow uses more than its fair share, for each active flow in the router. FRED penalizes flows with high strike values, i.e., unresponsive flows which fail to respond to congestion notification. Due to the per-flow state required at each router, FRED approach does not scale well.

CSFQ is a more scalable approach than FRED as it maintains per flow state only at the edge routers. CSFQ views the Internet as islands of routers, where flows enter the island through special routers called edge router and all the routers within the island are called core routers. In CSFQ, edge routers compute the per flow rate estimate by keeping per flow states and label the packets passing through them by inserting these estimates into each packet header. The core routers do not have to keep any per flow state, and employ a probabilistic dropping algorithm based on the flows arrival rate specified in the packet labels and an estimate of the aggregate traffic maintained by the core routers. Since the flow's outgoing rate is the minimum of the incoming rate and its fair share at the core routers, the core routers in CSFQ are capable of updating the labels in the packets with the new rate estimate.

Both these approaches only deal with throughput fairness amongst the flows and do not consider the relative importance of queuing delays to different applications represented by these flows. An interactive multimedia application would rather sacrifice some throughput from its fair share to receive lower queuing delays, whereas a throughput-sensitive application such as FTP would rather accept higher queuing delays than lower throughput. Thus, if a router can allow the applications with different requirements to trade throughput or delay with each other, then it can improve the overall performance of all applications.

The two AQM mechanisms presented in this thesis, try to give equal importance to throughput and delay requirements of an application. The first mechanism, RED-Worcester, tries to meet the average requirements of the incoming packets in terms of throughput and delay, whereas second mechanism, RED-Boston, tries to provide customized delay-throughput service to all applications.

2.3 Class Based Approaches

Class based approaches do consider varying QoS requirements of applications and provide different classes of service from which applications can choose.

Alternate Best Effort (ABE) [HKBT01] provides two classes of service, blue for throughput-sensitive applications and green for delay-sensitive applications. ABE uses two queues, a lager queue for blue traffic and a smaller queue for green traffic and uses a deadline based scheduler to serve the packets from these queues. Thus, ABE allows delay-sensitive applications to sacrifice throughput for lower delays without affecting throughput-sensitive applications. However, ABE provides just two classes of service, thus restricting the applications to be either delay-sensitive or throughput-sensitive without allowing them to choose their own delay-throughput tradeoff.

Class Based Thresholds (CBT) [PJS99] provides class-based treatment with gurantees of bandwidth limits for different classes. For this, CBT tracks the queue occupancy for each class as well as the overall queue occupancy. Whenever a packet associated with a particular class arrives, the class's average queue occupancy is updated. The class average is then compared with the thresholds for the class and the drop decision is made as in RED. Thus, in CBT, each class's average queue occupancy does not exceed its allocated share of the queue. However, these classes are pre-defined and are fixed, and do not allow the applications to select their own delay-throughput tradeoffs. DCBT with ChIPs [CC00] extends CBT by providing dynamic thresholds for classes and lower jitter for multimedia traffic. But, DCBT also suffers from the same problem as of CBT, that is restricting the applications to pre-defined classes. In addition, both CBT and DCBT provide only per class differentiated throughput service and do not really provide any differentiated delay service.

DiffServ approaches such as Assured Forwarding (AF) [HBWW99] and Expedited Forwarding (EF) [JNP99] try to provide QoS within the current best effort Internet. DiffServ combines flows with similar requirements into traffic aggregates and then provides differentiated service to these traffic aggregates. In DiffServ, clients or organizations negotiate long term service agreements with their Internet Service Providers(ISPs) for specific classes of traffic and ISPs try to provision their networks for providing the requested service as long as the traffic is within its profile as specified by the agreement. There is no performance guarantee, but there are relative service guarantees between different classes. Providing differentiated services is difficult when traffic travels across different ISP domains as the ISPs also need to negotiate service agreements with each other. Diff-Serv also requires traffic monitors, markers, traffic shapers, classifiers and droppers and an overall framework which makes them work together. Due to the complicated process of agreement negotiations and monitoring traffic, it is not easy to provide as many classes as required by the large number of applications supported by the Internet without causing additional overhead.

Our first AQM mechanism, RED-Worcester, is a simple extension to ARED and does not provide any class based service. RED-Worcester provides the service corresponding to average requirements of the incoming packets. Thus, it dynamically tunes the router to provide better overall QoS than ARED. Our second AQM mechanism, RED-Boston, provides a continuum of classes for applications to choose from. RED-Boston does not confine applications to pre-defined classes and tries to meet their individual requirements. RED-Boston is more scalable as it preserves the best effort service and does not require traffic monitoring or shaping.

2.4 Integrated Services

The Integrated Services (IntServ) approach provides per flow QoS guarantees by explicitly reserving buffers and bandwidth at each router along the flow's path. IntServ requires an end-to-end signaling protocol such as RSVP [Wro97] for reserving resources before the flow is admitted into the network. Each router in the flow's path then applies an admission control algorithm to determine if there are sufficient resources to meet the requested service without affecting the existing flows. The flow is admitted only if all the routers along the flows path have sufficient resources to meet the flow's service requirements.

Once a flow is admitted, each router is then responsible for ensuring that the flow receives its requested service. For this it has to deviate from the best effort service and use a complicated scheduler which uses per flow service information maintained at the routers for scheduling packets. The IntServ architecture defines two major classes of service, Guaranteed Service and Controlled-Load service. The Guaranteed Service provides firm bounds on the service in terms of delays and throughput throughout the network. The Controlled-Load service does not give any quantitative guarantees about performance. However, it ensures that a flow will receive a quality of service closely approximating the QoS that the same flow would receive from an unloaded network element.

The IntServ approach is not scalable and incremental deployment is not possible as all the routers in the flow's path have to implement integrated services to make it effective. Also, the admission control process becomes very complicated in the presence of dynamic routing changes, which is very common in the current Internet.

Similar to Controlled Load service, RED-Worcester and RED-Boston, do not provide any absolute performance guarantee, but try to provide improved QoS support to applications within best effort Internet. As RED-Worcester and RED-Boston, do not require per flow reservations at each router, they are more scalable than IntServ approaches.

2.5 Queue Law

Figure 2.3 depicts the queue law as explained in [FB00], assuming a single congested router with uniform dropping probability. As the drop rate at the router increases the average queue size and hence, average queuing delays experienced by the incoming packets decreases. However, an increase in drop rate also means reduced throughput. Thus, the average queue size at the router decides the throughput and delay treatment given to flows passing through it, and both throughput and delay can be controlled by changing drop

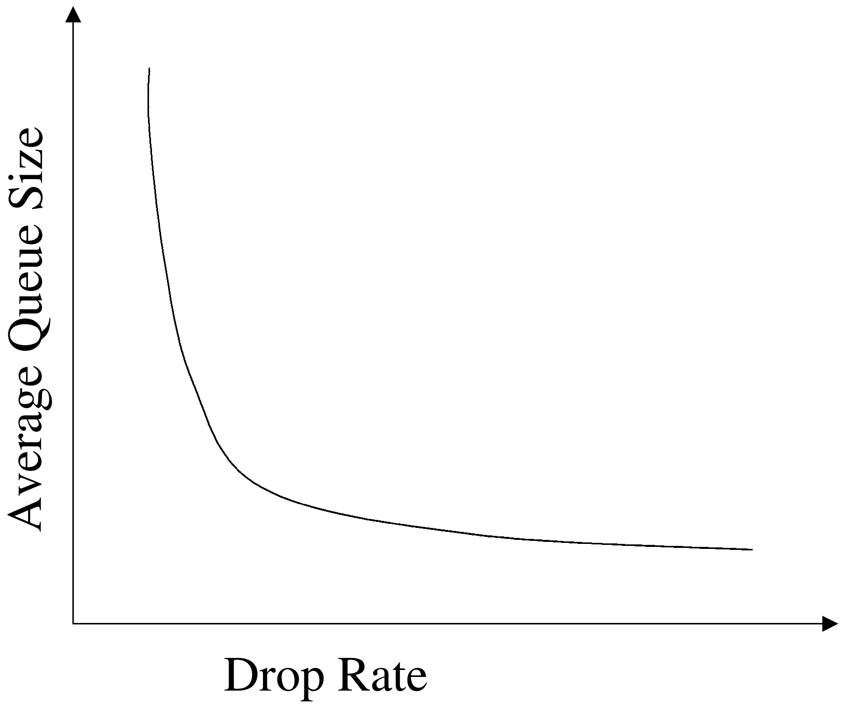


Figure 2.3: Queue Law

rate. ARED tries to maintain fixed average queue size, thus providing predictable average queuing delays by adapting drop rate. However, a fixed average queue size does not suit all kinds of traffic mixes and hence, the two AQM mechanisms proposed in this thesis, RED-Worcester and RED-Boston, adjust average queue size based on average requirements of the incoming traffic in order to provide better overall QoS at the router.

Chapter 3

RED-Worcester

This chapter explains the *RED-Worcester* mechanism and presents evaluation results with analysis. RED-Worcester tries to improves the overall QoS support at the router.

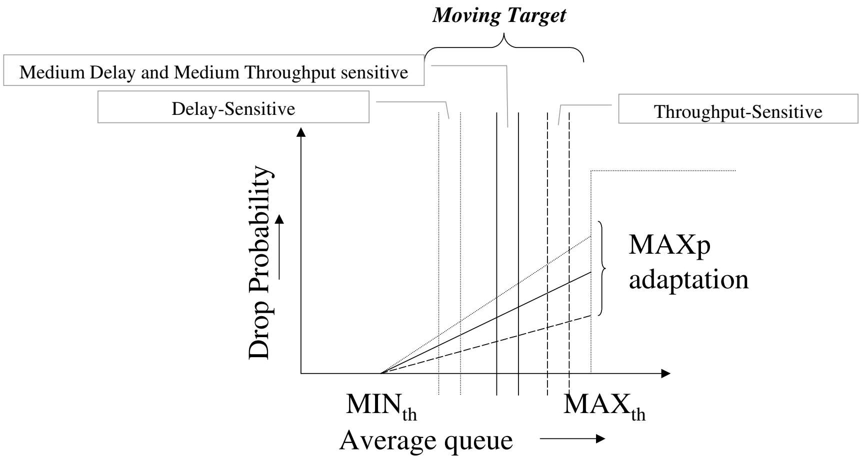


Figure 3.1: RED-Worcester

Unlike ARED's fixed target queue size, *RED-Worcester* updates its target queue size based on incoming traffic requirements as depicted in Figure 3.1. Thus, when incoming traffic is mostly throughput-sensitive, *RED-Worcester* maintains a higher average queue to improve the overall throughput. On the other hand, when incoming traffic is mostly delay-sensitive, *RED-Worcester* lowers the average queue size to reduce the average queuing delays. In case of medium throughput and medium delay sensitive traffic, *RED-Worcester* maintains an average queue size which gives medium throughput and delay performance.

RED-Worcester uses *delay hints* as explained in Section 3.1 to determine the requirements of incoming traffic in terms of delay and throughput at the router. The RED-Worcester router mechanism is described in Section 3.2. Section 3.3 discusses the overhead involved in RED-Worcester mechanism and Section 3.4 presents the evaluation of RED-Worcester with detail analysis of the results.

3.1 Delay Hints

In general, *source hints* are packet labels sent by end hosts or edge routers at the ingress to a network that carry information about a flow such as round trip time, bandwidth used, protocol type or other attributes. In RED-Worcester, application end-hosts indicate their sensitivity to queuing delays as source hints that we refer to as *delay hints*. The delay hint is not an absolute bound on queuing delay, but is rather a suggestion to Internet routers as to the relative importance of delay versus throughput. If delay hints are put in packet headers by edge routers, then existing applications can be easily supported without modification, where as if delay hints are inserted by the end hosts, then applications will have more flexibility in choosing their own hints. For flows which do not provide delay hints, RED-Worcester uses a default delay hint which corresponds to ARED's target queue size.

RED-Worcester does not guarantee queuing delays based on the delay hint, but uses the delay hint to calculate the average queuing requirements of incoming packets and adjust the average queue accordingly.

3.2 Mechanism

RED-Worcester router mechanism is a simple extension to ARED mechanism. RED-Worcester replaces the fixed target of ARED by a *moving target* as described in Section 3.2.1

3.2.1 Moving Target

For each incoming packet at the RED-Worcester router, the target queue size that a packet can tolerate is calculated based on the delay hint specified in the packet as shown below:

$$Target = C \times D/P \tag{3.1}$$

where C is the link capacity in bps, D is the delay hint in seconds and P is the packet size in bits. The *moving target* is calculated as the exponentially weighted average of the tolerable target queue size for each incoming packet as shown below:

$$Moving\,Target = (1 - w_t) \times Moving\,Target + w_t \times Target \tag{3.2}$$

where w_t is the weight used for calculating the exponentially weighted moving average. In our implementation, we used $w_t = 0.02$. Calculating the target in this way, ensures that average queue size at RED-Worcester router reflects the average requirements of the incoming packets.

3.3 Overhead

To use the QoS services provided by RED-Worcester, packets have to be labeled with delay hints. Based on discussion in [SZ99], there are from 4 to 17 bits in the IP header that may be available to carry source hints. Using milliseconds as the units for delay hints, 10 bits covers queuing delays from 0 to 1023 ms. Since 10 ms granularity is more than sufficient for most applications, the number of bits needed can be reduced to 7. The labeling of delay hints itself can be done either by end hosts, most likely the applications, or by edge routers at the ingress of a network. The RED-Worcester router has to read the labels from the incoming packets, an overhead similar to that in other approaches such as ABE [HKBT01], AF [HBWW99] and CBT [PJS99]. RED-Worcester calculates the moving queue average target, a calculation similar to the average queue calculation in ARED.

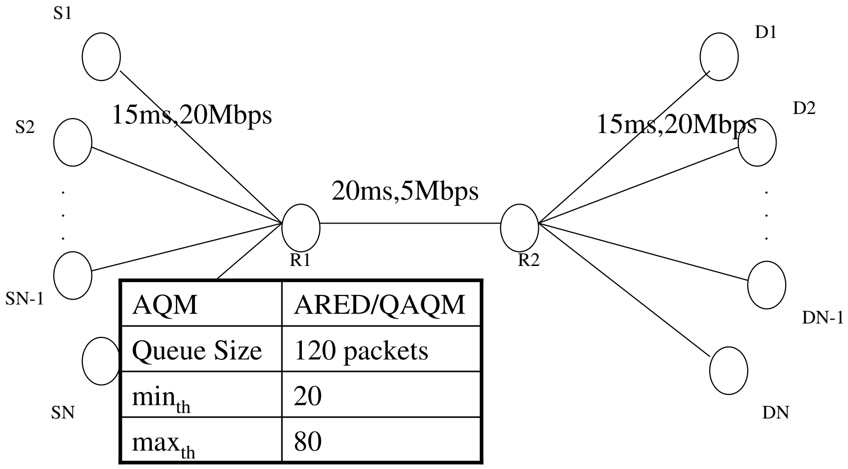
Thus, RED-Worcester adds a little complexity to ARED and in return improves the overall QoS support at the router.

3.4 Evaluation

This section presents the evaluation results of RED-Worcester with detailed analysis. In Section 3.4.1, the experimental setup and NS-2 [oCB] simulation details associated with our comparison study of the performance of RED-Worcester versus ARED are described. Section 3.4.2 presents our results and analysis.

3.4.1 Simulation Setup

This section describes the experimental setup and NS-2 simulation details associated with our comparison study of the performance of RED-Worcester versus ARED. The NS-2 simulator provides the ability to simulate drop-tail, RED and ARED routers. Additionally, NS-2 includes code to simulate both TCP protocols and TCP-friendly rate controlled protocols such as TFRC [FHPW00]. RED-Worcester was simulated by extending the ARED implementation and TCP NewReno and TFRC were modified to send delay hints.



AQM	ARED/QAQM
Queue Size	120 packets
min_{th}	20
max_{th}	80

Figure 3.2: Network Topology

The generic network topology used for all experiments is shown in Figure 3.2. S1-SN are traffic sources and D1-DN are destinations. The S-D pairs were varied to provide traffic mixes that included different combinations of TCP NewReno and TFRC flows, where the TFRC flows are meant to represent TCP-friendly multimedia flows. All sources send fixed-length 1000-byte packets. All links connecting sources to router R1 and all links connecting destinations to router R2 have a 20 Mbps capacity and a 15 ms delay. With the bandwidth and delay of the bottleneck link going from R1 to R2 set at 5 Mbps and 20 ms respectively, this topology is such that packet drops only occur at R1, where all measurements are made. For both RED-Worcester and ARED, the queue size at the congested router is 120 packets and min_{th} and max_{th} are set to 20 packets and 80 packets, respectively. All simulated flows start at time 0 sec and end at time 100 sec. Performance measurements are taken during the stable interval between 20 sec and 80 sec to avoid transient conditions from startup and stopping.

We first evaluate how RED-Worcester's target adjusts to changing traffic requirements. We plot average queue size under different traffic mixes to see that it does reflect the average requirements of the incoming flows and we also compare overall drop rates in

RED-Worcester with ARED. We measure queuing delays in milliseconds at the congested router.

However, analyzing throughput and delay alone is not sufficient as application performance realistically involves tradeoffs between throughput and delay. We propose a new metric, *QoS*, that provides a quantitative performance metric designed to capture to some degree the nature of the throughput-delay tradeoff. The correct choice of QoS function depends upon the the application's sensitivity to delay and throughput. An application may dynamically adjust its delay hints to current network conditions in attempt to improve its QoS. To include a wide spectrum of individual application types that can select their own tailored QoS objective, we provide a QoS measure that is generic while permitting customized combinations of delay and throughput measures for each individual applications:

$$QoS = T^{\alpha}/D^{\beta} \tag{3.3}$$

where T is throughput in bps, D is queueing delay in seconds at the router and $\alpha + \beta = 1$. While the choices of α and β depend upon the throughput and delay tolerances of applications themselves, the expectation is that the relativity of the delay hints from RED-Worcester enabled sources will vary accordingly. Note our new metric, QoS, is in fact a traffic-sensitive variant of the power performance metric.

In the experiments discussed in this thesis, $\alpha = 1$, is used for throughput-sensitive flows (such as Email or file transfer). Since these flows can tolerate delay, QoS for them depends only upon throughput. We use $\alpha = 0.5$ and $\beta = 0.5$ for delay-sensitive flows (such as an interactive videoconference), since these flows require low delay, but also moderate amounts of throughput. For medium throughput and medium delay-sensitive flows (such as some HTTP flows for Web browsing), we use $0.5 < \alpha < 1$ and $0 < \beta < 0.5$.

3.4.2 Analysis

This section presents the simulation results and their analysis for RED-Worcester and ARED. The simulations evaluate performance under a traffic mix that varies from mostly throughput-sensitive flows to mostly delay-sensitive flows. Figure 3.3 provides details on the traffic mixes used for our simulations.

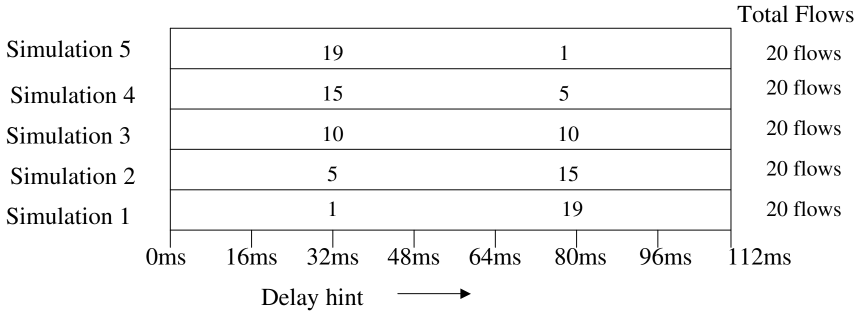


Figure 3.3: Simulation RED-Worcester and ARED: Traffic Mix

Each simulation in this set ran 20 flows using the network topology shown in Figure 3.2. Throughput-sensitive flows were represented by TCP flows carrying a delay hint of 80 ms, which corresponds to a queue size of 50 packets at the bottlenecked router R1. Delay-sensitive flows were represented by TFRC flows carrying a delay hint of 32 ms, which corresponds to a queue size of 20 packets at R1. The X-axes for all the graphs in this section indicate the changing traffic mix corresponding to the five different simulations described in Figure 3.3.

Figure 3.4 shows RED-Worcester's *moving target* as incoming traffic mix changes. When most of the incoming flows are throughput-sensitive carrying a delay hint of 80 ms, i.e., queue size of 50 packets, RED-Worcester's moving target settles at around 48 packets. As the number of delay-sensitive flows carrying a delay hint of 32 ms, i.e., queue size of 20 packets, increases, the moving target moves down linearly and finally settles at around 22 packets, when most of the flows are delay sensitive. The bars show the standard deviations in the moving target.

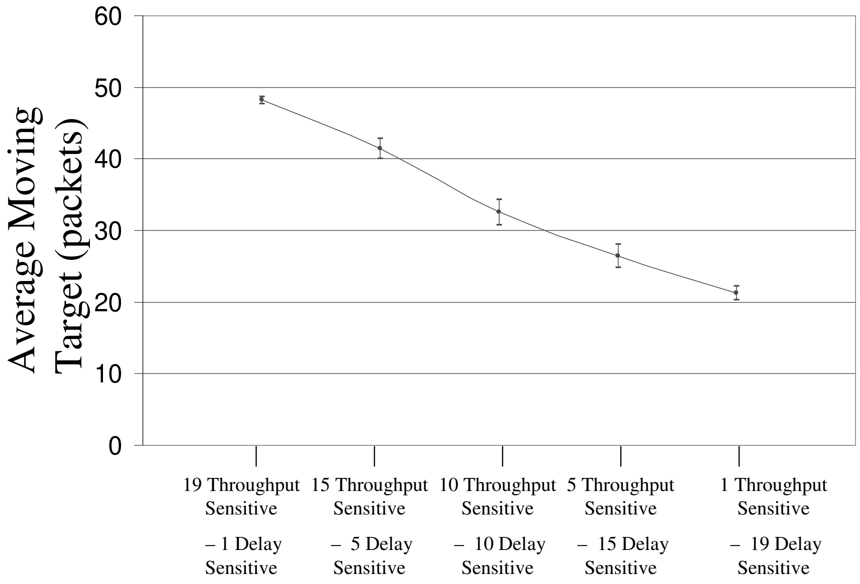


Figure 3.4: RED-Worcester's Moving Target

Figure 3.5 shows the max_p adaptation over time in RED-Worcester as the traffic mix changes from throughput-sensitive (low drop probability) to delay-sensitive (high drop probability). When most of the incoming flows are throughput-sensitive, max_p settles to a comparatively lower value, allowing the average queue size to grow. This improves the overall throughput performance of the RED-Worcester router, allowing it to serve the throughput-sensitive flows in a better way. On the other hand, when most of the incoming flows are delay-sensitive, max_p settles to a comparatively higher value, thus forcing RED-Worcester router to maintain a smaller average queue size. This reduces the average queuing delays in the router, thus allowing it to serve the delay-sensitive flows in a better way. When the perce-

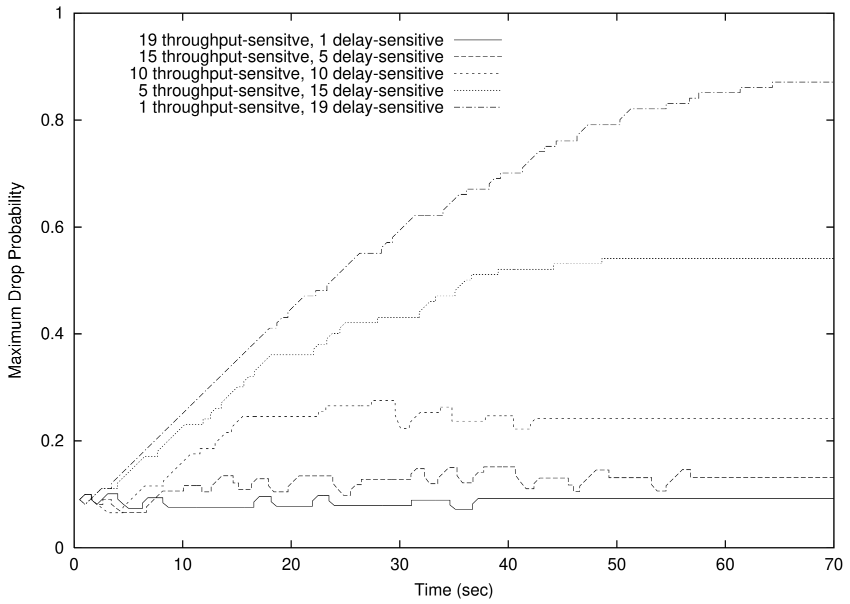


Figure 3.5: max_p Analysis I

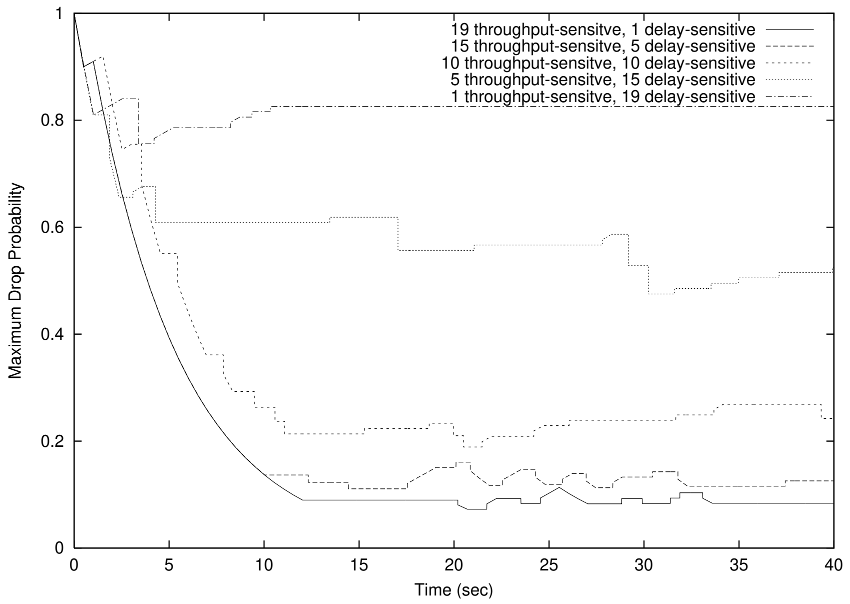


Figure 3.6: max_p Analysis II

ntages of throughput-sensitive and delay-sensitive flows in the incoming traffic are equal, max_p settles to a value such that the average queue size at the router will be of medium size. Thus, the RED-Worcester router makes a compromise, when there are equal number of throughput-sensitive and delay-sensitive flows and gives a medium delay and medium throughput service. Figure 3.6 shows the max_p adaptation with initial value of max_p fixed to 1. Due to AIMD approach of max_p adaptation, reduction in max_p value is much quicker than increase in max_p value.

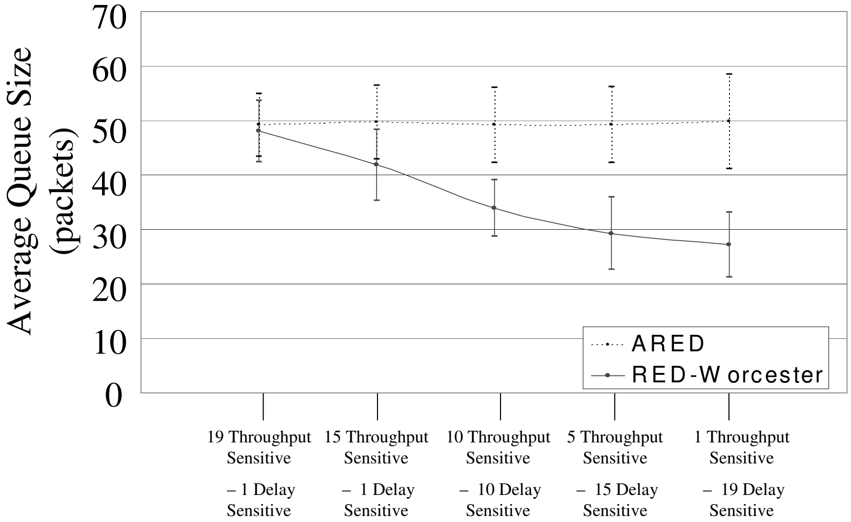


Figure 3.7: Average queue size in RED-Worcester

Figure 3.7 shows the average queue size measured in packets for RED-Worcester and ARED. ARED maintains a fixed average queue size half way between min_{th} and max_{th} irrespective of the incoming traffic mix. In our simulations, min_{th} is 20 packets and max_{th} is 80 packets, so ARED's fixed target is at 50 packets. However, RED-Worcester, owing to its moving target, adapts average queue size based on the incoming traffic requirements. Thus, when the incoming traffic has higher fraction of throughput-sensitive flows, RED-Worcester maintains a higher average queue size, and when the incoming traffic has higher fraction of delay-sensitive flows, it maintains a lower average queue size.

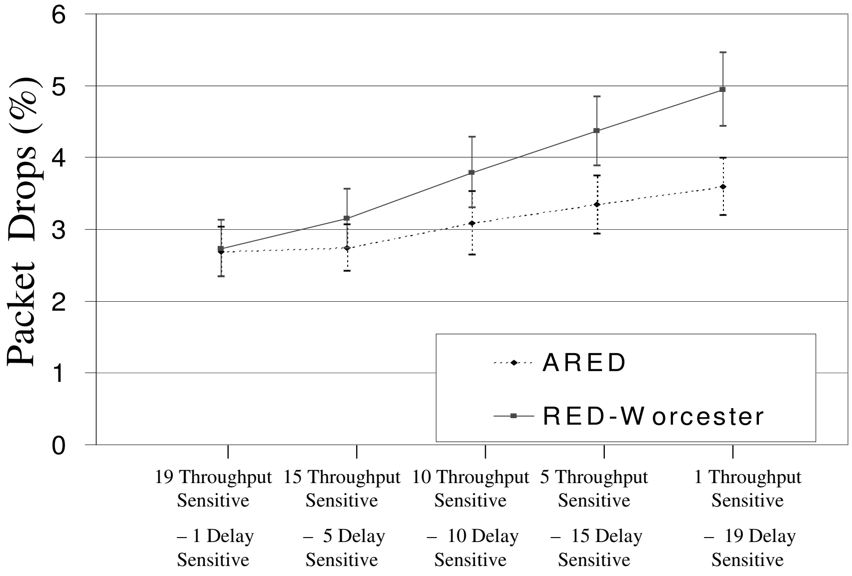


Figure 3.8: Percent packet drops

Figure 3.8 shows the average per flow percent packet drops in RED-Worcester and ARED. In case of RED-Worcester, the percent packet drop rate increases linearly as the traffic mix changes from mostly throughput-sensitive traffic to mostly delay-sensitive traffic. This is because of the RED-Worcester's moving target which shifts to smaller average queue sizes as the fraction of delay-sensitive flows increases in the incoming traffic mix, thus causing max_p to adapt to higher values. Surprisingly, even in ARED, as the fraction of delay-sensitive flows increases, the drop rate goes up. This suggests that when the total number of flows is constant and TCP flows are replaced by TFRC flows, ARED has to increase its dropping rate to maintain the same average queue size. This suggest's that TFRC flows are not exactly TCP friendly and may sometimes use more bandwidth than TCP flows under the same situation.

Figure 3.9 shows the QoS performance of throughput-sensitive flows in RED-Worcester. The QoS for throughput-sensitive flows is calculated as $QoS = T^1/D^0$, since throughput-sensitive flows can tolerate queuing delays. The performance of throughput-sensitive

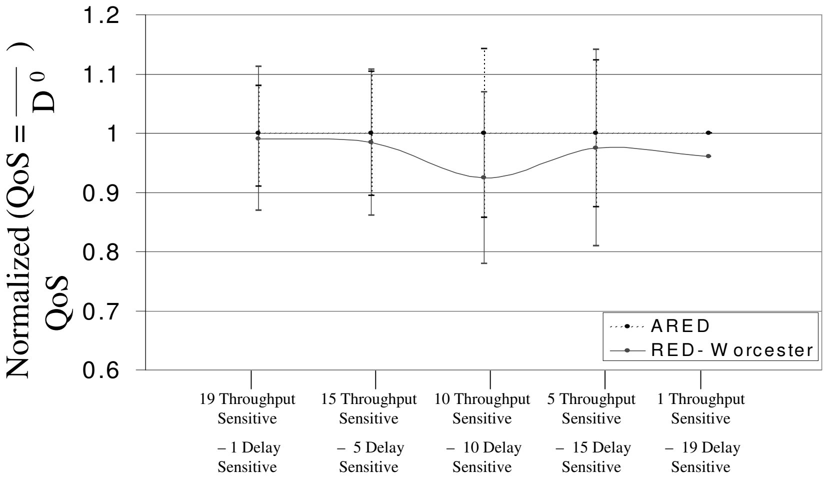


Figure 3.9: Normalized QoS for Throughput-Sensitive Flows

flows suffers slightly in RED-Worcester as compared to ARED. The throughput-sensitive flows in our simulations used a delay hint corresponding to a queue size of 50 packets, which is also the ARED's fixed target queue size. Since RED-Worcester adapts its target based on incoming traffic requirements, RED-Worcester's target queue size is always below ARED's fixed target in the presence of delay-sensitive flows, thus giving lower throughput performance. Note that if the throughput-sensitive flows had used higher delay hints than ARED's target queue size, then their QoS performance would have been better in RED-Worcester than in ARED, when throughput-sensitive flows are in majority in the incoming traffic mix. RED-Worcester does not favor any particular type of flows, but instead tries to meet the average requirements of the incoming traffic. ARED, on the other hand, due to its fixed target, gives comparativley better performance to throughput sensitive flows than delay sensitive flows.

Figure 3.10 shows the normalized QoS for delay-sensitive flows in RED-Worcester and ARED calculated as $QoS = T^{0.5}/D^{0.5}$. QoS for delay-sensitive flows improves in RED-Worcester as the fraction of delay-sensitive flows increases in the incoming traffic mix. RED-Worcester's moving target brings the average queue size down as the fraction of delay-sensitive flows increases, reducing the overall average queuing delays. On the

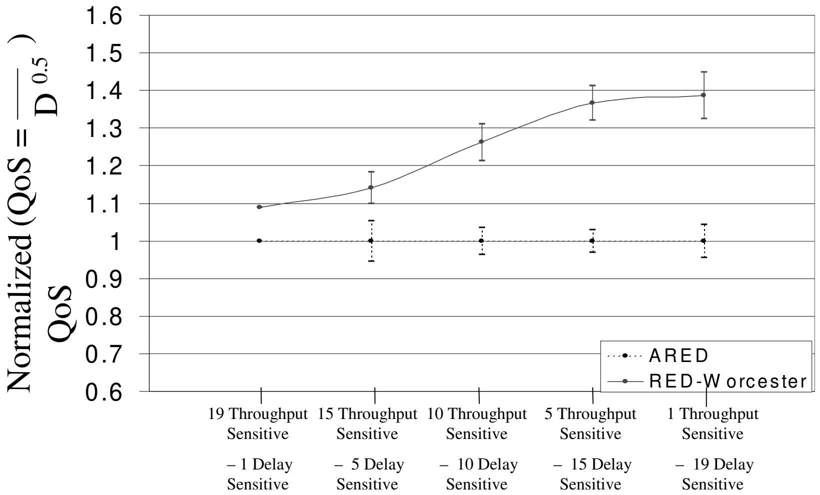


Figure 3.10: Normalized QoS for Delay-Sensitive Flows

contrary, in ARED, delay-sensitive flows suffer as ARED maintains the average queue size at fixed target of 50 packets irrespective of incoming traffic.

Our evaluation shows that RED-Worcester router has a capability of tuning itself to meet the average throughput and delay requirements of the incoming flows, although it cannot provide exact delay-throughput tradeoff to individual flows as per their requirements.

Chapter 4

RED-Boston

This chapter presents implementation details of RED-Boston along with its evaluation. The goal of RED-Boston is to provide improved support for varying application delay-throughput requirements while fitting in with the current best-effort Internet environment. Applications specify their individual requirements and receive QoS more tailored to their requirements, without requiring additional charges or policing mechanisms.

As in RED-Worcester, in RED-Boston applications mark each outgoing packet with delay hints. However, with RED-Boston applications can derive much more benefit by adapting delay hints as explained in Section 4.1. The router mechanism itself is explained in Section 4.2. In RED-Boston, packets experience a drop probability based on their delay hint and average drop probability, as explained in Section 4.2.1 while being inserted in the outgoing queue based on their delay hint relative to the average delay hint, as explained in Section 4.2.2. The overhead and complexity of the RED-Boston service is discussed in Section 4.3 and Section 4.4 presents the results of our evaluation with detail analysis.

4.1 Delay Hints

Delay hints are explained in Section 3.1. This Section explains how applications can adapt delay hints dynamically to receive improved performance from RED-Boston routers.

At the RED-Boston router, a relatively low delay hint indicates an application's desire for lower delay while a higher delay hint suggests the application prefers higher throughput even if this implies higher delays. Applications can dynamically adapt their delay hints based on observed throughput and latency. Thus applications can use delay hints to provide significant input into the delay-throughput tradeoff decisions made at the router.

Applications such as FTP or Email that want to minimize overall transfer time without significant concern over individual packet delays would choose a high delay hint. Applications such as multimedia streaming that seek to minimize end-to-end delay could choose a low delay hint. However, under congested network conditions these applications may discover that the low delay hint also yields unacceptably low throughput. Applications could then raise their delay hints until measured throughput reached acceptable levels. Thus, sources are able to vary their delay hints in any manner that increases their perceived QoS. Packets that carry no delay hints are handled using a default delay hint that corresponds to the router's target queue size.

Figure 4.1 presents a simple illustration of how a delay-sensitive TCP-friendly application might use delay hints. An interactive videoconference running over a company T1 link competes with n TCP flows for the 1.5 Mbps of link bandwidth. The videoconference requires a minimum data rate, R, of about 384 Kbps[1] to insure acceptable video quality. This rate is depicted by the horizontal dotted line in Figure 4.1. Under these conditions, the videoconference is free to choose the delay hint. The curved dashed lines in Figure 4.1 depict the approximate bandwidth an application would receive for delay hints of 20 ms, 40 ms, 60 ms and 80 ms (reading left to right). The higher the delay hint, the

[1] A typical minimum data rate for an H.261 videoconference.

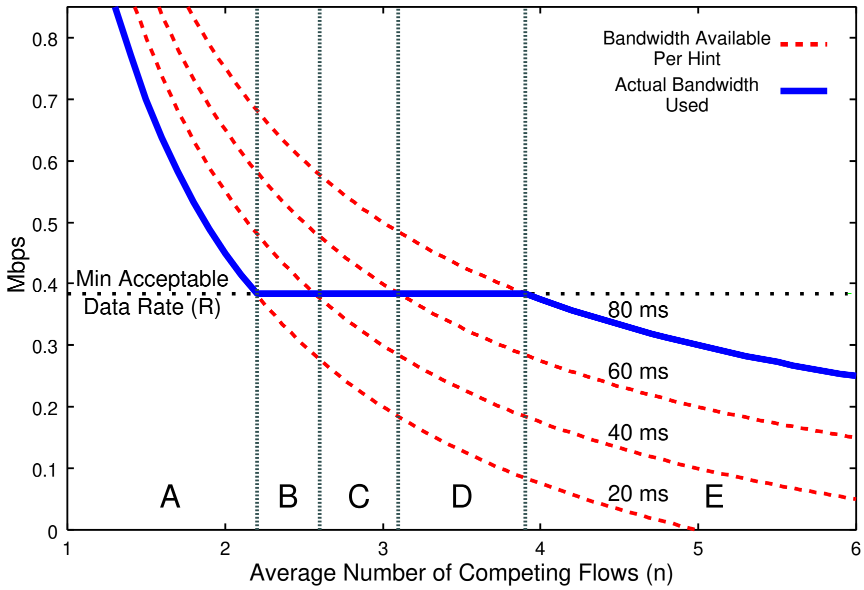


Figure 4.1: Possible Hint Strategy for an Interactive Multimedia Application

more bandwidth the videoconference will receive, but the higher the delay. The lower the delay hint, the less bandwidth the videoconference will receive, but the lower the delay. The "best" delay hint for the application depends upon the perceived quality, pq, for each throughput and delay combination. For this simplified example, we assume that pq is 0 when the data rate is less than R and increases inversely with the delay for $pq > R$. In other words, once the videoconference obtains its minimum data rate, the greatest benefit to perceived quality comes from lower delay. Under these conditions, the vertical lines in Figure 4.1 indicate regions *A*, *B*, *C*, *D* and *E* where the videoconference would dynamically adapt delay hints. In region *A*, the videoconference receives the required minimum bandwidth by using the lowest delay hint, 20 ms. As the T1 link became more congested, the videoconference would increase delay hints gradually from 20 ms to 40 ms in region *B* and from 40 ms to 60 ms in region *C* to maintain the minimum acceptable data rate. In region *D*, as the level of congestion increases, the videoconference is forced to increase the delay hints further upto 80 ms, to obtain the minimum bandwidth. This is the same delay hint as is likely used by the competing TCP flows. In region *E*, even with a large delay hint, the available bandwidth drops below the minimum acceptable rate, in which case the videoconference user would probably terminate the connection. Note that this example is greatly simplified and we would expect perceived quality functions that trade-off throughput and latency in a more sophisticated manner would be used.

4.2 Mechanism

The RED-Boston router mechanism directly extends RED-Worcester. Arriving delay hints affect three aspects of the RED-Boston router mechanism. First, since it uses RED-Worcester, the target average queue size moves to adjust the average queue size as described in Section 3.2.1. Second, the delay hint determines the drop probability, where higher delay hints mean lower drop probabilities and vice versa, as explained in Section 4.2.1. Third, the delay hint also determines where the incoming packet is placed in the outgoing queue as described in Section 4.2.2.

4.2.1 Delay Hint Based Drop Probability

Since RED-Boston provides different queuing delays for different delay hints, the drop probability must be adjusted based on delay hint. RED-Boston calculates the average drop probability p based on the average queue size like ARED, but adjusts the per-packet drop probability p' based on the delay hint and the average delay as follows:

$$p' = p \times delay/delay\_hint \tag{4.1}$$

where $delay$ is the queuing delay corresponding to the average queue size. The moving target of RED-Worcester allows RED-Boston to maintain the average queue size at approximately the average of the aggregate traffic's delay requirements, providing fairness in the per-packet drop probability calculation. RED-Boston limits the drop probability to be 0.5 or less, as we speculate there cannot be any meaningful QoS with drop rates higher than 50%.

4.2.2 Weighted Insert

The RED-Boston outgoing queue is sorted based on packet weights. An incoming packet is inserted in the sorted queue based on its weight which is computed as shown below:

$$weight = arrival\ time + delay\_hint \tag{4.2}$$

Arrival time is used as an aging mechanism in this calculation to avoid packet starvation. Under normal conditions a flow's delay hints will not vary and differences in arrival times will prevent packet reordering. Since packets are served from the front of the sorted queue, the packet with the lowest weight will be served first. Note the delay hint is not an upper bound on queuing delay for the packet but rather a relative hint such that packets with lower delay hints experience lower delays than packets with higher delay hints. However, our results show that on average, packets receive a queuing delay close to the delay hint specified.

Thus, RED-Boston's *weighted insert* provides delay-sensitive packets with a relatively lower queuing delay. On the other hand, RED-Boston's *delay hint based drop probability* compensates by providing delay tolerant packets with a lower drop probability and hence higher throughput.

4.2.3 Algorithm

Figure 4.2 summarizes the RED-Boston algorithm. Each packet contains a delay hint, as described in Section 4.1. For each incoming packet, the target average queue size is updated as in RED-Worcester. If there is extreme congestion, indicated by the queue average being above max_{th}, the packet is dropped. If there is congestion, as indicated by the queue average being between min_{th} and max_{th}, the drop probability for the packet is computed based on the queue parameters, the $delay\_hint$, and the average queuing delay, $delay$, as described in Section 4.2.1. If the packet is not dropped, it is inserted in the queue based on a weighting of its arrival time and delay hint, as described in Section 4.2.2.

4.3 Overhead

RED-Boston incurs all the overhead of RED-Worcester as discussed in Section 3.3. In addition to this, RED-Boston's weighted insert mechanism is slightly more complex than ABE's duplicate scheduling with deadlines mechanism, but provides much more flexibility. The weighted insert mechanism can be implemented using a probabilistic data structure such as skip lists [Pug90], giving complexity $O(log(n))$.

The overall complexity of RED-Boston is much less than IntServ approaches and is comparable to class based approaches, while providing better QoS service than typical class based approaches. Unlike DiffServ approaches, RED-Boston does not require negotiation of service level agreements and does not required traffic monitoring and shaping, and integrates more easily into the current best-effort Internet environment.

4.4 Evaluation

For evaluating RED-Boston, we used the same simulation setup as that of RED-Worcester evaluation, described in Section 3.4.1. This section presents results and analysis from two sets of NS-2 simulations designed to compare RED-Boston's performance against ARED. Section 4.4.1 focuses on the first set of simulations consisting of different percentages of

on receiving packet pkt:
// Calculate moving target
// (see Section 3.2.1)
$target$ = $(1 - w_t)\times target$ + $w_t\ C\ \times$
$\quad pkt.delay\_hint/(pkt.size \times 8)$

if (q_{avg}>=max_{th}) then

dropPacket(pkt, 1)

elseif (avg>=min_{th}) then

// Calc drop prob p*, based on RED*
// (see Section 2.1)
p=calcDropP(q_{avg},min_{th},max_{th},max_p)

//Calc delay hint based drop prob, p'
//(see Section 4.2.1)
p'=$p\times(delay/delay\_hint)$

if (NOT dropPacket(pkt,p')) then

// Insert in the queue based on weight
// (see Section 4.2.2)
$weight$=$arrival\ time$+$pkt.delay$
insertPacket(pkt,$weight$)

Every *interval* seconds:
// Adjust max_p *so* q_{ave} *hits target*
// (see Figure 2.2)
max_p= adaptMaxP(max_p,$target$,q_{avg})

Figure 4.2: RED-Boston Algorithm

delay-sensitive and throughput-sensitive flows in the incoming traffic mix. Section 4.4.2 focuses on the second set of simulations consisting of incoming traffic with a range of throughput and delay requirements.

4.4.1 Simulation Set - 1

The objective of the first set of simulations is to evaluate RED-Boston's robustness as the incoming traffic mix at the router varies from mostly throughput-sensitive traffic to mostly delay-sensitive traffic. Figure 4.3 provides details on the traffic mixes used for the first five simulations.

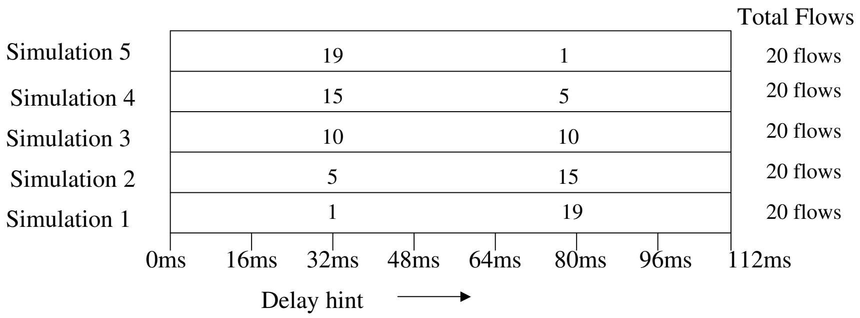


Figure 4.3: Simulation Set-1: Traffic Mix

Each simulation in this set ran 20 flows using the network topology shown in Figure 3.2. Throughput-sensitive flows were represented by TCP flows carrying a delay hint of 80 ms, which corresponds to a queue size of 50 packets at the bottlenecked router R1. Delay-sensitive flows were represented by TFRC flows carrying a delay hint of 32 ms which corresponds to a queue size of 20 packets at R1. The X-axis for all the graphs in this section indicate the changing traffic mix corresponding to the five different simulations described in Figure 4.3.

Figure 4.4 shows the average queue size in packets for RED-Boston and ARED as the traffic mix varies. While the average queue size remains nearly constant across all the ARED simulations, the moving target mechanism in RED-Boston allows the average queue size to adjust to the delay hint distributions caused by changes in

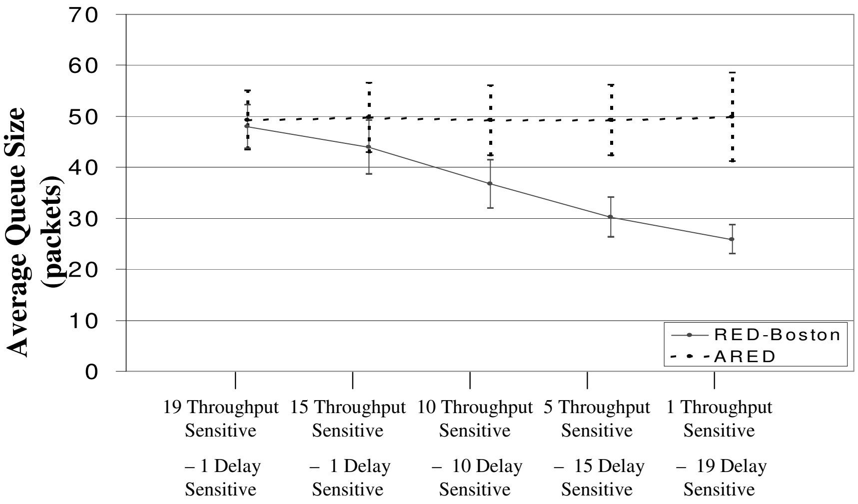


Figure 4.4: Simulation Set-1: Average Queue Size

the incoming traffic's QoS requirements. Thus, when traffic is mostly throughput-sensitive, the average queue size is higher to increase the overall throughput. Correspondingly, when traffic is mostly delay-sensitive, the average queue size becomes smaller to reduce the overall queuing delays. The RED-Boston average queue sizes in these five experiments reflects the average requirements of incoming traffic. On the other hand, Adaptive RED's target range is insensitive to incoming traffic's requirements.

Figure 4.5 shows the queuing delays experienced by the throughput-sensitive and the delay-sensitive flows in simulation set 1. For ARED, graphing only the average delay curve is necessary because the throughput-sensitive and delay-sensitive flows receive identical treatment from ARED despite having different QoS requirements. Figure 4.5 indicates that RED-Boston gives lower queuing delays to delay-sensitive flows as compared to throughput-sensitive flows. Moreover, when the delay-sensitive flows represent a small percentage of the incoming traffic, the queuing delays experienced by the delay-sensitive flows is close to the 32 ms delay hint. However, as the percentage of delay-sensitive flows increases in the incoming traffic mix, the average queuing delay for these flows increases slightly. This is caused by many delay-sensitive flows competing with each

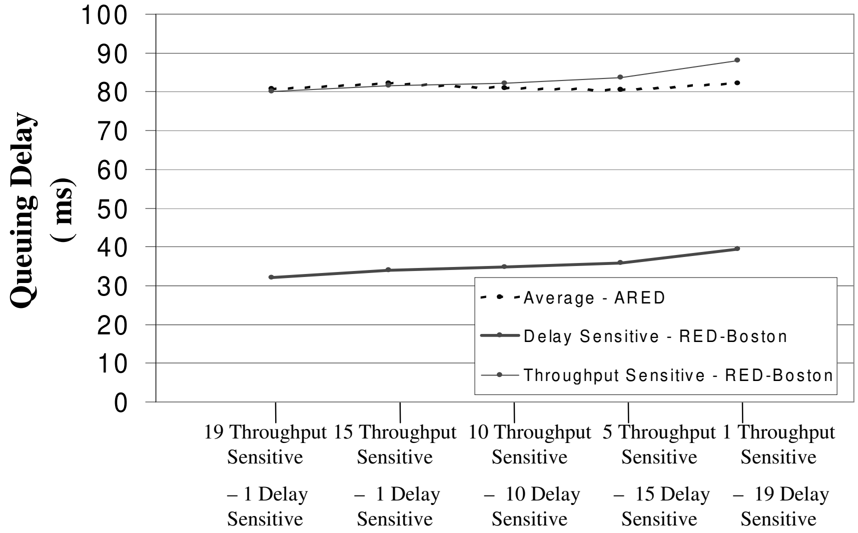


Figure 4.5: Simulation Set-1: Queuing Delays

other for lower delays. Conversely, when most of the incoming flows are throughput-sensitive, their average queuing delay is near the 80 ms delay hint. As the percentage of delay-sensitive flows increases in the incoming traffic, the average queuing delay for throughput-sensitive flows increases slightly. This increase happens because RED-Boston lets the delay-sensitive flows cut in the queue ahead of throughput-sensitive flows to get lower delays. However, from Figure 4.5, it is clear that the aging component of the RED-Boston packet weight calculation prevents starvation of throughput-sensitive flows.

Figure 4.6 shows the average percentage of packets dropped per flow in RED-Boston and ARED. Again only one curve is needed for ARED, since ARED treats all flows equally regardless of the QoS requirements. For RED-Boston, Figure 4.6 does separate out the average per flow percentage of packets dropped for throughput-sensitive flows and delay-sensitive flows. The delay sensitive flow competing with 19 throughput-sensitive flows has to pay a high drop rate for the low delay service it receives because it is inserted far below the average queue size determined by the throughput sensitive flows. As the fraction of delay-sensitive flows increases, the average per flow drop rates for the delay-sensitive flows decreases. This is because as the number of delay sensitive flows increases,

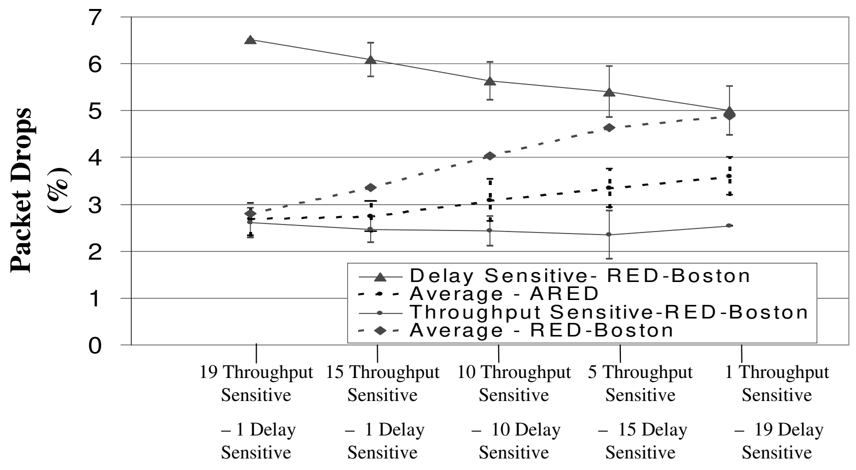


Figure 4.6: Simulation Set-1: Average Per Flow Percent of Packets Dropped

RED-Boston's moving target brings the average queue size down to give a lower delay service and the corresponding drop rate is shared equally by all the delay-sensitive flows. For throughput-sensitive flows, the RED-Boston drop rate is consistently lower than that of ARED and is almost constant for all simulations irrespective of the number of delay-sensitive flows.

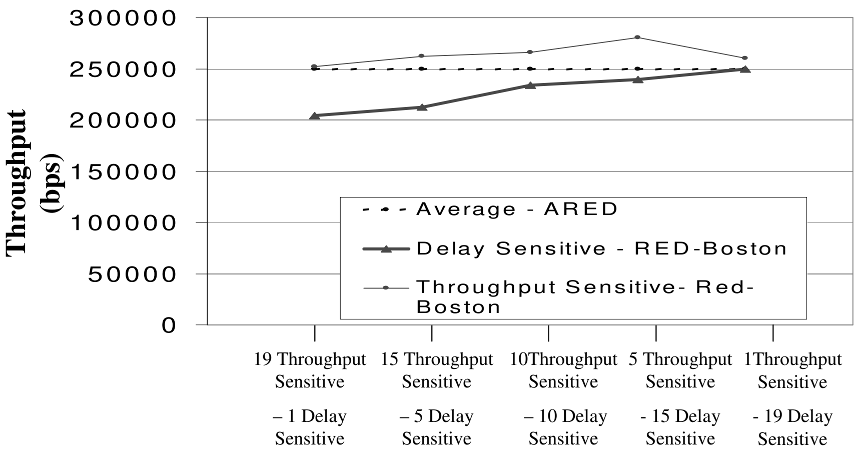


Figure 4.7: Simulation Set-1: Average Per Flow Throughput

Figure 4.7 shows the average per flow throughput for ARED and average per flow throughput for throughput-sensitive flows and delay-sensitive flows for RED-Boston. The ARED throughput is nearly constant for all simulations and is the same for all flows. In

RED-Boston, delay-sensitive flows get lower throughput than throughput-sensitive flows in exchange for low delay service. When there are 19 throughput-sensitive and one delay-sensitive flow, the delay-sensitive flow sacrifices some throughput for lower delay. This additional throughput sacrificed by one delay-sensitive flow is equally shared by the 19 throughput-sensitive flows. However, as the number of delay-sensitive flows increases, their throughput penalty decreases since the average queue size is reduced. The throughput sacrificed by delay-sensitive flows is shared equally by the existing throughput-sensitive flows in each case. In practice, as described in Section 4.1, if the reduced throughput given to a delay-sensitive flow falls below an acceptable threshold for that flow, the flow can send a larger delay hint if it is able to trade higher delay for higher throughput.

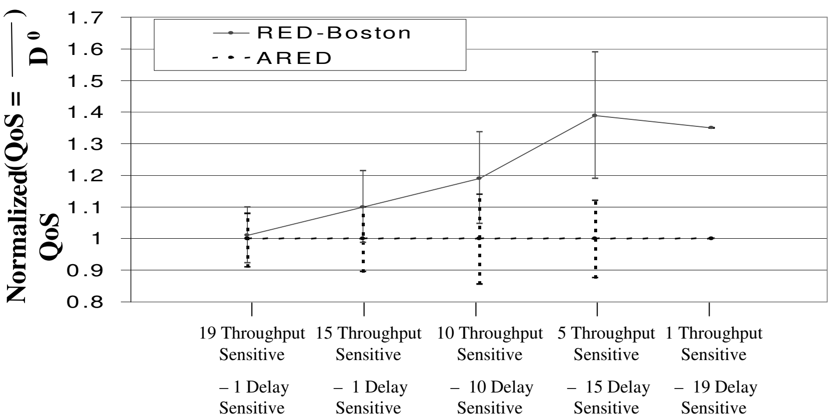


Figure 4.8: Simulation Set-1: Normalized QoS for Throughput-Sensitive Flows

Figure 4.8 presents the QoS for throughput-sensitive flows traversing a RED-Boston router normalized with respect to the QoS for throughput-sensitive flows traversing an ARED router. As described in Section 3.4.1, QoS for throughput-sensitive flows is defined as $QoS = T^1/D^0$. The RED-Boston throughput-sensitive flows record consistently higher QoS than the equivalent ARED flows. Furthermore, the RED-Boston QoS advantage grows as the percentage of delay-sensitive flows in the traffic mix increases. When delay-sensitive flows dominate the mix, more flows are willing to give up throughput for lower delay and, as a result, the throughput sensitive flows experience higher QoS even though they are in the minority.

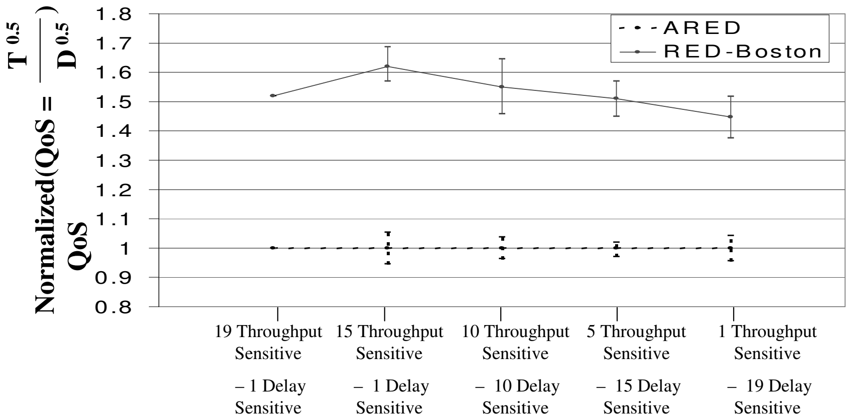


Figure 4.9: Simulation Set-1: Normalized QoS for Delay Sensitive Flows

Paralleling Figure 4.8, Figure 4.9 graphs the QoS for delay-sensitive flows in RED-Boston normalized with respect to the QoS for delay-sensitive flows in ARED. As described in Section 3.4.1, the specific QoS used for delay-sensitive flows is $QoS = T^{0.5}/D^{0.5}$. Using this metric, RED-Boston yields a QoS improvement (on average) for delay-sensitive flows more than 50 percent over delay-sensitive flows served by an Adaptive RED router. The QoS for delay-sensitive flows in RED-Boston decreases as the number of delay-sensitive flows increases, because they start competing with each other for low delay service. The resulting improvement seen in the QoS for the delay sensitive flows captures the improvement in the quality that interactive multimedia applications using TFRC would experience with RED-Boston routers.

4.4.2 Simulation Set - 2

The objective of the second set of simulations was to evaluate RED-Boston with a range of throughput and delay requirements. Each simulation in this set ran 35 TCP flows using the network topology shown in Figure 3.2. Delay hints for these TCP flows were uniformly varied from 16 ms to 112 ms in 16 ms increments, as depicted in Figure 4.10.

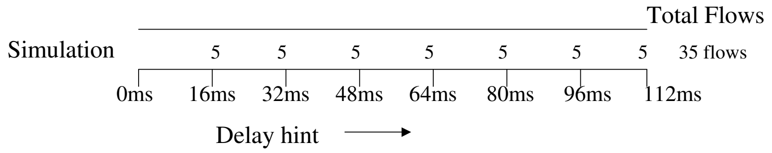


Figure 4.10: Simulation Set-2: Traffic Mix

This flow distribution was used for simulation of a congested RED-Boston router which is compared against simulation with the same flow distribution for a congested Adaptive RED router. The X-axis for all the graphs in this section indicate the flows with the 16 ms to 112 ms hints described in Figure 4.10.

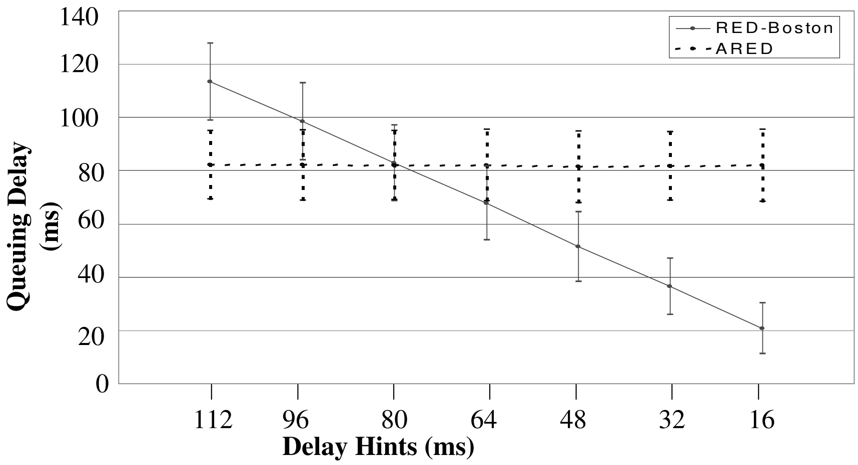


Figure 4.11: Simulation Set-2: Queuing Delays

Figure 4.11 shows the queuing delays in milliseconds for ARED and RED-Boston. While ARED provides an essentially identical average delay of 80 ms for all delay hint flow groups, the average delay seen by flow groups served by RED-Boston linearly decreases as the delay-sensitivity of the flows increases. Moreover, RED-Boston provides average queuing delays comparable to the delay hints specified by the flows in each of the seven delay hint groups.

Figure 4.12 shows the differences in the percentage of packet drops for ARED and

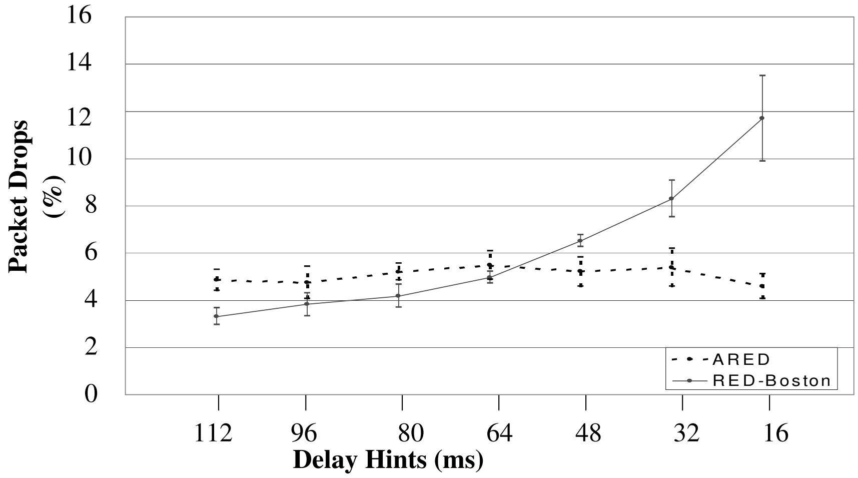


Figure 4.12: Simulation Set-2: Average Per Flow Percentage of Packets Dropped

RED-Boston. ARED yields an average drop percentage near 5% for all the delay-hint flow groups. However, in the RED-Boston simulation, the average packet drop percentage per delay-hint flow group increases as the delay sensitivity of the flows increases. Delay sensitive flows, such as interactive audio, would typically incorporate repair techniques [PHH98] to counter act the higher loss rates.

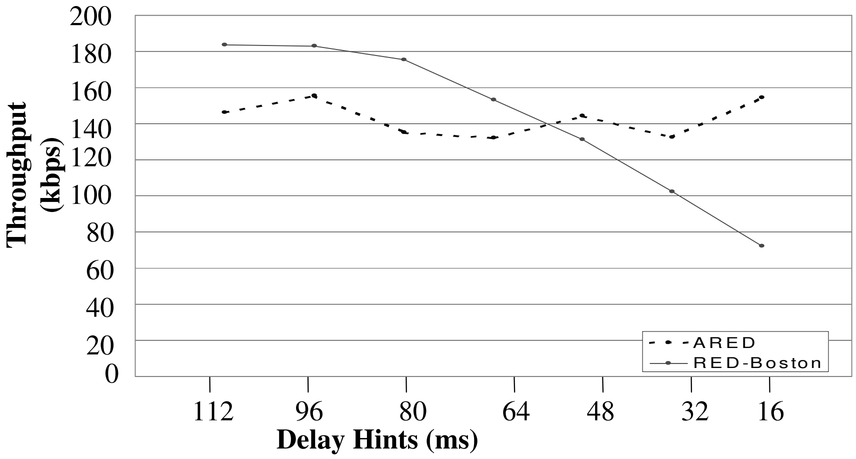


Figure 4.13: Simulation Set-2: Average Per Flow Throughput

Figure 4.13 shows the average per flow throughput for each flow group for ARED and RED-Boston. For ARED, the throughput averages around 140 Kbps for all flow groups. Contrastingly, for RED-Boston as the delay sensitivity increases the lower source hints yield significantly decreased average throughput. This demonstrates clearly that RED-Boston facilitates the trading of lower throughout for lower delay that interactive multimedia flows would be willing to make.

To evaluate the impact of RED-Boston's differentiated service to flows carrying different delay hints, the QoS metric associated with the seven distinct delay-hint flow groups in simulation set 2 is adjusted by linearly interpolating α and β in the QoS formula, as shown in Table 4.1.

$$QoS = T^{\alpha}/D^{\beta}$$

Delay Hint (ms)	α	β
112	1	0
96	0.917	0.083
80	0.834	0.166
64	0.75	0.25
48	0.67	0.33
32	0.584	0.416
16	0.5	0.5

Table 4.1: Simulation Set-2: QoS Parameters

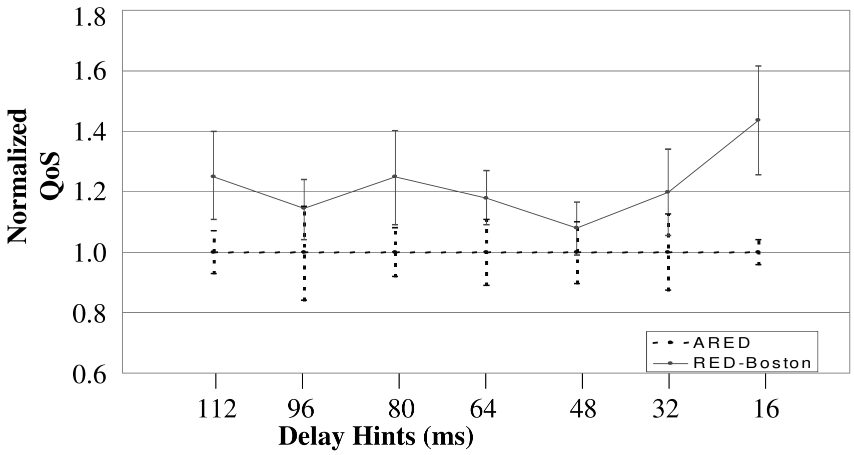


Figure 4.14: Simulation Set-2: Normalized QoS

Figure 4.14 shows the average QoS results for the seven flow groups with different delay hints in the set 2 simulations. The RED-Boston QoS values are normalized against the ARED QoS values. This figure demonstrates that RED-Boston provides higher QoS to all flows regardless of where a flow is on the delay-throughout sensitivity spectrum.

Thus, our evaluation of RED-Boston shows that RED-Boston is capable of providing applications approximately their desired delay-throughput tradeoff with the current best effort service architecture.

Chapter 5

Conclusions

Applications supported by the current Internet have varying requirements in terms of throughput and delays. On one end, traditional applications such as FTP and E-mail are throughput-sensitive and can tolerate considerable delays, whereas on the other hand interactive multimedia applications such as online games are delay-sensitive and can sacrifice some throughput for low delay service. However, current Internet routers give equal treatment to all flows without considering the applications specific or even aggregated requirements. This can result in performance degradation of the applications, if routers are not tuned properly to meet their requirements.

In this thesis, we have presented two active queue management approaches to improve the quality of service support provided to the applications by the routers. In our proposed approaches, applications indicate their relative delay-throughput sensitiveness to the routers through *delay hints* and routers adjust their operating parameters to provide better service to applications.

Our first mechanism, RED-Worcester, is a simple modification to Adaptive RED to improve the overall QoS provided by the router. RED-Worcester maintains an exponentially weighted moving average of the delay hints specified in the incoming packets to estimate the avrage queuing requirements of the flows passing through it. Based on this information, RED-Worcester adjusts its average queue size to match the average delay requirements of the flows. Thus, when most of the incoming flows are throughput-sensitive, RED-Worcester maintains a higher average queue size to provide better throughput, whereas when most of the incoming flows are delay-sensitive, RED-Worcester mainatins a lower average queue size to provide low delay service. When there is no majority of any particular type of traffic, RED-Worcester maintains a average queue size to meet average requirements of the flows. Our evaluation of RED-Worcester router shows that it consistently adapts its average queue size under changing traffic conditions to meet the average requirements of the flows.

Our second mechanism, RED-Boston, further extends RED-Worcester and tries to meet the individual requirements of each incoming flow. RED-Boston uses weighted insert to insert the incoming packet in the outbound queue based on its delay hint and arrival time, and uses a delay hint based drop probability to apply higher dropping rates to relatively delay-sensitive packets over throughput-sensitive packets. Thus, RED-Boston provides low delay service to delay-sensitive flows in exchange for lower throughput and high throughput service to throughput-sensitive flows in exchange for higher delays. The applications can choose the exact delay-throughput tradeoff as the delay spectrum is continuous. Applications even have the flexibility of adapting their delay hints based on observed performance. Our evaluation of RED-Boston shows that RED-Boston gives better QoS to delay-sensitive as well as throughput-sensitive applications under varying traffic mixes than does ARED under similar situations. When the incoming traffic mix consists of applications with varying requirements, RED-Boston consistently gives better QoS to all applications as compared to ARED.

Both the mechanisms discussed in this thesis preserve the best-effort nature of the current Internet and do not require any traffic monitoring or charging mechanisms.

Chapter 6

Future Work

RED-Boston expects flows to be responsive to network congestion in a TCP-friendly fashion. RED-Boston needs to be evaluated for more richer traffic mix including UDP flows. Since RED-Boston provides delay hint based drop rate and queuing delays, unresponsive flows should not gain more of a bandwidth advantage under RED-Boston than they do under current Internet environments. Still, an extension to RED-Boston would be to enhance it with a rate based active queue management technique such as CSFQ [SSZ98], as described in Section 2.2. When CSFQ support is added to RED-Boston, the dropping probability can be calculated based on combination of a flow's rate and its delay hint. Thus, responsive flows can be protected from the unresponsive flows and flows with same delay hint can be assured a similar bandwidth share.

Another extension to RED-Boston could be an interaction between cascaded RED-Boston routers. Applications could provide queuing delay hints as explained in Section 3.1 and each RED-Boston router could do an additional task of updating the cumulative amount of time a packet has waited in router queues. Thus, down stream RED-Boston routers would have additional information of total queuing delay experienced by a packet and its relative delay hint, which can be used by the router to make better delay-throughput tradeoff decisions.

Another possible future work is to develop applications that make use of RED-Boston to improve their performance. Such applications can use perceived quality functions to evaluate their own performance dynamically and adjust the delay hints accordingly.

Bibliography

[BFPT99] J-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-Based Error Control for Internet Telephony. In *Proceedings of IEEE INFOCOM*, March 1999.

[CC00] Jae Chung and Mark Claypool. Dynamic-CBT and ChIPS - Router Support for Improved Multimedia Performance on the Internet. In *Proceedings of the ACM Multimedia Conference*, November 2000.

[CJOS00] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith. Tuning RED for Web Traffic. In *Proceedings of ACM SIGCOMM Conference*, August 2000.

[FB00] V. Firoiu and M Borden. A study of active queue management for congestion control. In *In Proceedings of the Conference on Computer Communications (IEEE infocom)*, March 2000.

[FF99] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, February 1999.

[FGS01] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Under submission, http://www.icir.org/floyd/papers/adaptiveRed.pdf, 2001.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of ACM SIGCOMM Conference*, pages 45 – 58, 2000.

[FJ92] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet Switched Gateways. Internetwroking:Research and Experience, V.3 N.3, 1992.

[FJ93] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[Has89] E. Hashem. Analysis of Random Drop for Gateway Congestion Control. Technical Report Report LCS TR-465, Laboratory for Computer Science, MIT, 1989.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *IETF Request for Comments (RFC) 2597*, June 1999.

[HKBT01] P. Hurley, M. Kara, J. Le Boudec, and P. Thiran. ABE: Providing a Low Delay within Best Effort. *IEEE Network Magazine*, May/June 2001.

[JNP99] V. Jacobson, K. Nichols, and K. Poduri. Expedited Forwarding PHB Group. *IETF Request for Comments (RFC) 2598*, June 1999.

[LC00] Yanlin Liu and Mark Claypool. Using Redundancy to Repair Video Damaged by Network Data Loss. In *Proceedings of IS&T/SPIE/ACM Multimedia Computing and Networking (MMCN)*, January 2000.

[LM97] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proceedings of ACM SIGCOMM Conference*, September 1997.

[MBDL99] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy red. In *Proc. of 7th International Workshop on Quality of Service(IWQoS'99)*, pages 260–262, June 1999.

[oCB] Universiy of California Berkeley. The Network Simulator - ns-2. Interent site
`http://www.isi.edu/nsnam/ns/`.

[OLW99] T. Ott, T. Lakshman, and L. Wong. Sred:stabilized red. In *Proceedings of IEEE, infocomm*, March 1999.

[PCM00] C. Padhye, K. Christensen, and W. Moreno. A New Adaptive FEC Loss Control Algorithm for Voice Over IP Applications. In *Proceedings of IEEE International Performance, Computing and Communication Conference*, February 2000.

[PHH98] Carlos Perkins, Orlin Hodson, and Vicky Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network Magazine*, Sep/Oct 1998.

[PJS99] Mark Parris, Kevin Jeffay, and F. Smith. Lightweight Active Router-Queue Management for Multimedia Networking. In *Proceedings of Multimedia Computing and Networking (MMCN), SPIE Proceedings Series*, January 1999.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balalnced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[PW99] K. Park and W. Wang. QoS-Sensitive Transport of Real-Time MPEG Video Using Adaptive Forward Error Correction. In *Proceedings of IEEE Multimedia Systems*, pages 426 – 432, June 1999.

[RHE99] Rezza Rejaie, Mark Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of IEEE Infocom*, 1999.

[SSZ98] Ion Stoica, Scott Shenker, and Hui Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM Conference*, September 1998.

[SZ99] Ion Stoica and Hui Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of ACM SIGCOMM Conference*, September 1999.

[TZ99] D. Tan and A. Zakhor. Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol. *IEEE Transactions on Multimedia*, May 1999.

[WCZ01] Yubing Wang, Mark Claypool, and Zheng Zuo. An Empirical Study of RealVideo Performance Across the Internet. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[Wro97] J. Wroclawski. The Use of RSVP with IETF Integrated Services. *IETF Request for Comments (RFC) 2210*, September 1997.