Traffic Sensitve Active Queue Management for Improved Multimedia Streaming

Vishal Phirke, Mark Claypool, Robert Kinicki
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609
{vish|claypool|rek}@cs.wpi.edu

ABSTRACT

The Internet, which has traditionally supported throughput-sensitive applications such as email and file transfer, is increasingly supporting delay-sensitive multimedia applications such as interactive audio. These delay-sensitive applications would often rather sacrifice some throughput for lower delay. Unfortunately, the current Internet does not offer choices in the amount of delay or throughput an application receives, but instead provides monolithic best-effort service to all applications. This paper proposes and evaluates a new Active Queue Management (AQM) technique that employs source hints to provide service at network routers that is sensitive to the Quality of Service (QoS) expectations for a variety of applications. Applications indicate their delay or throughput sensitivity via a delay hint in their outgoing packets. The router, which we call RED-Boston, uses the delay hints to dynamically adjust the router to yield better delay performance for delay-sensitive applications and better throughput for throughput-sensitive applications. Using a new QoS metric, our simulations demonstrate that RED-Boston yields higher QoS than Adaptive RED for both throughput-sensitive flows and delay-sensitive flows. RED-Boston operates equally well in all traffic scenarios and fits the current best-effort Internet environment without requiring traffic monitoring.

1. INTRODUCTION

The Internet today carries traffic for applications with a wide range of delay and loss requirements. Traditional applications such as FTP and E-mail are primarily concerned with throughput, while Web traffic is moderately sensitive to delay as well as throughput. 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 [1, 11, 13, 14] 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.

Unfortunately, current Internet routers are not able to provide a choice in Quality of Service (QoS).[1] Current and proposed router queue mechanisms can be classified as in Figure 1 on the basis of the level of QoS support provided and the complexity of the router implementation.

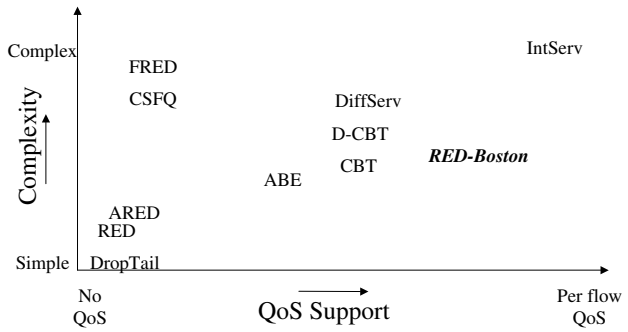

Figure 1: Approaches for QoS in Internet Routers

Due to the simplicity of the FIFO queuing mechanism, drop-tail queues are the most widely used queuing scheme in Internet routers today. When drop-tail buffers overflow, newly arriving packets are dropped regardless of the application type of the arriving 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. This best-effort service provides no consideration for multimedia flows that can be severely affected by high delays. Clearly, drop-tail provides very limited, if any, QoS support.

RED [6], 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 through-

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

put. Adaptive RED (ARED) [4] 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 [22]. However, UDP is unresponsive to packet drops that indicate congestion at the router. FRED [10] and CSFQ [19] 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* [3]. Thus, in the near future all multimedia applications are likely to use TCP-friendly protocols that adjust their transmission rates smoothly, such as in [5, 18, 21] 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 [8] 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 [15] 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 [2] 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) [7] and Expedited Forwarding (EF) [9], 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 [23] by all routers along a connection on a per-flow basis, making scalability difficult for global deployment.

We present a new AQM approach for IP networks that provides QoS service for applications with a variety of delay and throughput requirements, such as file transfer, Web browsing and interactive audio. Unlike approaches that provide fixed classes of service, each application sending traffic into the RED-Boston router chooses a customized delay-throughput trade-off based on its own requirements. The service is still best-effort in that it requires no additional policing mechanisms, charging mechanisms or usage control. Under the proposed service, RED-Boston routers operate equally well under scenarios with only traditional traffic and operate better under scenarios with mixed or multimedia only traffic.

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. At a congested router, the queue manager, which we have named *RED-Boston*[2], uses the delay hints of all packets to calculate an average delay hint and a target average queue size chosen to best fit the aggregate traffic. RED-Boston uses the target average queue size to compute an average drop probability similar to that in ARED [4]. Unlike ARED where all packets are either enqueued at the tail of the queue or dropped based on the same drop probability, RED-Boston inserts packets in the outbound queue based on their delay hint relative to the average delay hint. Packets with a delay hint lower than the average delay hint are inserted towards the front of the queue, while being dropped with a higher than average drop probability. Conversely, packets with a delay hint higher than the average delay hint are inserted towards the back of the queue, while being dropped with a lower than average drop probability. Thus, using a low delay hint, delay-sensitive applications experience reduced end-to-end delay, while receiving the lower throughput resulting from an increased drop rate. Similarly, using a high delay hint, throughput-sensitive applications receive an increased throughput resulting from a decreased drop rate, while having an increased end-to-end delay.

RED-Boston is evaluated via simulation under a wide variety of traffic mixes. The results show that RED-Boston achieves better overall application performance for both delay-sensitive multimedia applications and throughput-sensitive applications. As traffic mix changes from mostly throughput-sensitive to mostly delay-sensitive, RED-Boston gives consistently better performance than ARED. When the traffic mix consists of applications with a range of delay and throughput requirements, RED-Boston achieves a performance that is more suitable to all applications than performance under ARED.

This paper is organized as follows: Section 2 explains the RED-Boston mechanism and provides an example of how an application may use the RED-Boston service; Section 3 describes simulation experiments used to evaluate RED-Boston; Section 4 analyzes the simulation results; and Section 5 contains conclusions and suggests possible future work.

2. RED-BOSTON

This section provides a detailed explanation of the RED-Boston scheme. The goal of this scheme is improved support for varying application delay-throughput requirements while fitting in with the current best-effort Internet environment.

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

Applications specify their individual requirements without requiring additional charges or policing mechanisms.

Applications mark each outgoing packet with a delay hint that suggests the relative importance of delay versus throughput for the packet. Delay hints are explained in Section 2.1. The router mechanism itself is explained in Section 2.2. The router uses the delay hints to calculate a target average queue size as explained in Section *2.2.2*. Packets are inserted in the outgoing queue based on their delay hint relative to the average delay hint, as explained in Section *2.2.4*, while receiving a drop probability based on their delay hint and average drop probability, as explained in Section *2.2.3*. The overhead and complexity of the RED-Boston service is discussed in Section 2.3.

2.1 Delay Hints

In general, *source hints* are packet labels sent by end hosts or edge router 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-Boston, application end-hosts indicate their sensitivity to queuing delay 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.

RED-Boston does not guarantee delay based on the delay hint, but the delay hint facilitates the router in providing an appropriate service. At the RED-Boston router, a relatively low delay hint indicates an applications desire for lower delay and lower throughput while a higher delay hint suggests the application prefers higher throughput even if this implies higher delays. Applications can dynamically adapt the delay hints based on observed throughput and latency. Since the spectrum of desired delay is continuous, delay hints provide applications with 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 to send 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 2 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 bandwidth. The videoconference requires a minimum data rate, R, of about 384 Kbps[3] to insure acceptable video quality. This rate is depicted by the horizontal dotted line in Figure 2. Under these conditions, the videoconference is free to choose the source hint. The curved dashed lines in Figure 2 depict the approximate bandwidth an application would receive for source hints of 20 ms, 40 ms, 60 ms and 80 ms (reading left to right). The higher the source hint, the more bandwidth the videoconference will receive, but the higher the delay. The lower the source hint, the less bandwidth the videoconference will receive, but the lower the delay. The "best" source 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 2 indicate regions A, B, C, D and E where the videoconference would choose different delay hints. In region A, the videoconference required minimum bandwidth can be obtained by using the lowest delay hint, 20 ms. As the T1 link became more congested, the videoconference would choose source hints of 40 ms and 60 ms for regions B and C respectively. In region D, the videoconference is forced to use the largest source hint, 80 ms, to obtain the minimum bandwidth. This is the same source hint as is likely used by the competing TCP flows. In region E, even with a large source 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.

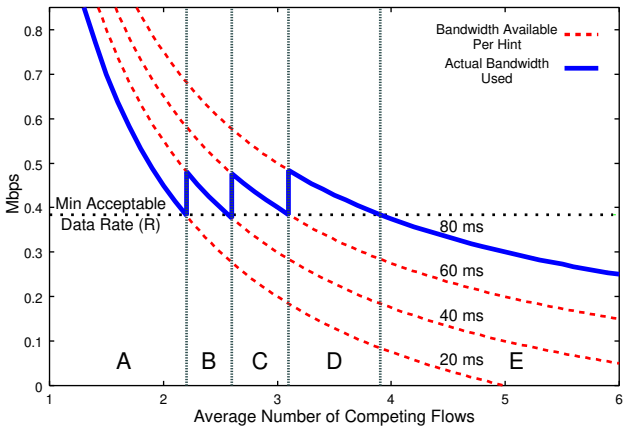

Figure 2: Possible Hint Strategy for an Interactive Multimedia Application

2.2 Mechanism

The RED-Boston router mechanism directly extends Adaptive RED (ARED) [4], as described in Section *2.2.1*, by using delay hints to tune the router for better overall QoS. Arriving delay hints affect three aspects of the RED-Boston router mechanism. First, the target average queue size moves to adjust the average queue size as described in Section *2.2.2*. Second, the delay hint determines the drop probability, where higher delay hints mean lower drop probabilities and vice versa, as explained in Section *2.2.3*. Third, the delay hint also determines where the incoming packet is placed in the outgoing queue as described in Section *2.2.4*.

2.2.1 Adaptive RED

ARED builds a target queue range on top of the RED [6] mechanisms. RED monitors the outgoing queue for impending congestion, accommodating transient congestion by

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

keeping an exponential weighted moving average of the queue (q_{ave}). When congestion is detected, indicated by the queue average rising above a fixed minimum threshold (min_{th}), packets are randomly dropped with a fixed drop probability for each packet. The probability of packet drop increases linearly from zero at the minimum threshold to a maximum drop probability (max_p) at the maximum threshold (max_{th}).

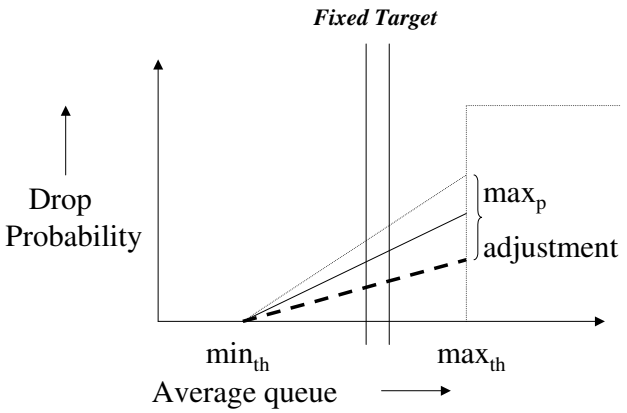

Figure 3: Adaptive RED

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 3. 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 4.

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 ×= β *// 0.9, by default*

Figure 4: Adaptive RED Algorithm for Adjusting Average Queue to the Target Queue

2.2.2 Moving Target

Adaptive RED [4] controls the average queuing delay by keeping q_{ave} within a target range. In ARED, the targetted average queue size is independent of traffic QoS requirements. We argue that the targetted average queue size should be based on the incoming traffic's requirements. RED-Boston extends ARED by providing a moving target queue size based on an exponentially weighted moving average of the delay hints of the incoming packets, as shown below:

$$target = (1 - w_t)target + w_t C \times delay_hint/(packet_size \times 8) \quad (1)$$

C is the router's outgoing link capacity and w_t is the weight for calculating the exponentially weighted moving average (set to $w_t = 0.02$ in our implementation). The weighted average allows the queue target to stabilize even if the individual packets arriving from different flows have different delay hints. When overall traffic changes from throughput-sensitive to delay-sensitive, the target reflects the average queuing delay requirements of the incoming packets.

2.2.3 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:

$$p' = p \times delay/delay_hint \quad (2)$$

where *delay* is the queuing delay corresponding to the average queue size. The moving target described in Section 2.2.2 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 maximum drop probability to be 0.5, as we speculate there cannot be any meaningful QoS with drop rates higher than 50%.

2.2.4 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:

$$weight = arrival\ time + delay_hint \quad (3)$$

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 earliest deadline 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.

2.2.5 Algorithm

Figure 5 summarizes the RED-Boston algorithm. Each packet contains a delay hint, as described in Section 2.1. For each incoming packet, the delay hint is used to update the moving target for average queue size, kept by the router, as described in Section 2.2.2. 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 2.2.3. 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 *2.2.4*.

on receiving packet *pkt*:
// Calculate moving target
// (see Section 2.2.2)
$target = (1 - w_t)\ target + w_t\ C \times$
$pkt.delay_hint/(pkt.size \times 8)$

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

dropPacket(*pkt*, 1)

elseif (avg>=min_{th}) then

// Calc drop prob p, based on RED
// (see Section 2.2.1)
p=calcDropP(q_{avg},min_{th},max_{th},max_p)

//Calc delay hint based drop prob, p′
//(see Section 2.2.3)
p'=$p \times (delay/delay_hint)$

if (!dropPacket(*pkt*,p')) then

// Insert in the queue based on weight
// (see Section 2.2.4)
$weight$=$arrival\ time$+$pkt.delay$
insertPacket(*pkt*,$weight$)

Every ***interval*** **seconds:**
// Adjust max_p so q_{ave} hits target
// (see Section 3)
max_p= adaptMaxP(max_p,$target$,q_{avg})

Figure 5: RED-Boston Algorithm

2.3 Overhead

To use the QoS services provided by RED-Boston, packets have to be labeled with delay hints. Based on discussion in [20], 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. As the example in Section 2.1 shows, the labeling of packets by applications gives them flexibility to adjust the hints based on measured levels of throughput and delay to maximize the application-specific QoS. However, if the edge routers label hints, legacy applications can be supported without change.

The RED-Boston router has to read the labels from the incoming packets, an overhead similar to that in other approaches such as ABE [8], AF [7] and CBT [15]. RED-Boston calculates the queue average target, a calculation similar to the average queue calculation in ARED.

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 [17], 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.

3. EXPERIMENTS

This section discusses the experimental methods and NS-2 simulation details associated with our comparison study of the performance of RED-Boston versus ARED. The NS-2 simulator [12] 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 [5]. RED-Boston was simulated by extending the ARED implementation with the *moving target*, *weighted insert* and *delay hint based drop probability* algorithms as explained in Section 2 and, TCP NewReno and TFRC were modified to send delay hints.

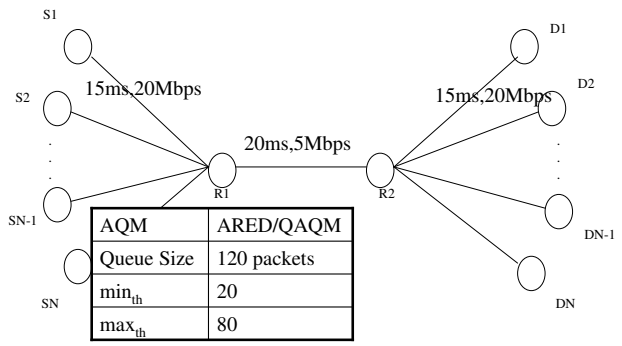

Figure 6: Network Topology

The generic network topology used for all experiments is shown in Figure 6. S1-SN are traffic sources and D1-DN are destinations. The S-D pairs were varied to provide traffic mixes that included different mixes of TCP NewReno and TFRC flows (see Section 4), 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-Boston 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.

To ascertain overall router performance, we measure link

utilization, average queue size and drop rate. Since RED-Boston is designed to be sensitive to per-flow throughput and delay concerns, average throughput (in bps) and delay (in ms) are measured for flows with different delay hints.

Moreover, since application performance realistically involves tradeoffs between throughput and delay, we believe analyzing either alone is not sufficient. 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. As illustrated in Section 2.1, 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 application:

$$QoS = T^{\alpha}/D^{\beta} \quad (4)$$

where $\alpha + \beta = 1$. While the choices of α and β will depend upon the throughput and delay tolerances of applications themselves, the expectation is that the relativity of the delay hints from RED-Boston 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 the next section, $\alpha = 1$, 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 $0 < \beta < 0.5$.

4. ANALYSIS

This section presents results analysis of two sets of NS-2 simulations designed to compare RED-Boston's performance against ARED. Section 4.1 focuses on the first set of simulations consisting of changing the percentages of delay-sensitive and throughput-sensitive flows in the incoming traffic mix. Section 4.2 focuses on the second set of simulations consisting of incoming traffic with a range of throughput and delay requirements.

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 7 provides details on the traffic mixes used for the first five simulations.

Each simulation in this set ran 20 flows using the network topology shown in Figure 6. 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 7.

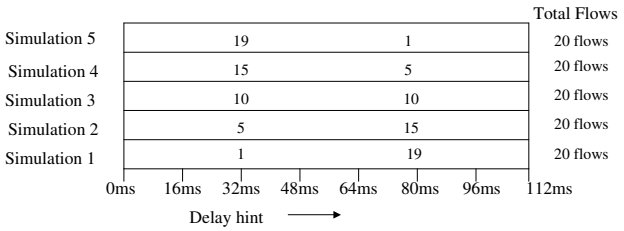

Figure 7: Simulation Set-1: Traffic Mix

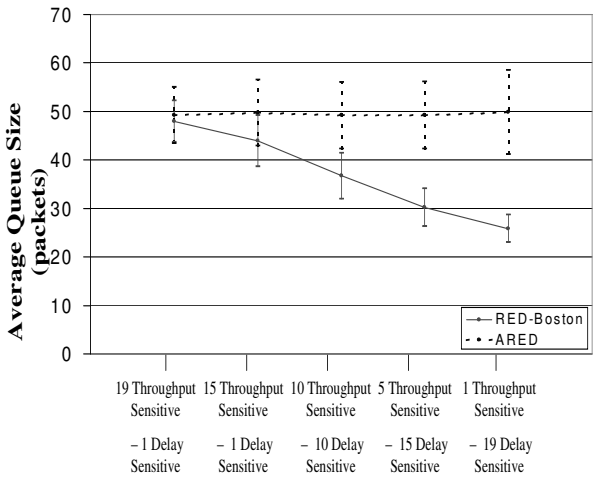

Figure 8: Simulation Set-1: Average Queue Size

Figure 8 shows the average queue size in packets for RED-Boston and ARED as the traffic mix varies. While the average queue size remains 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 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 9 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 9 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 other for lower delays. Conversely, when most of the incoming flows are

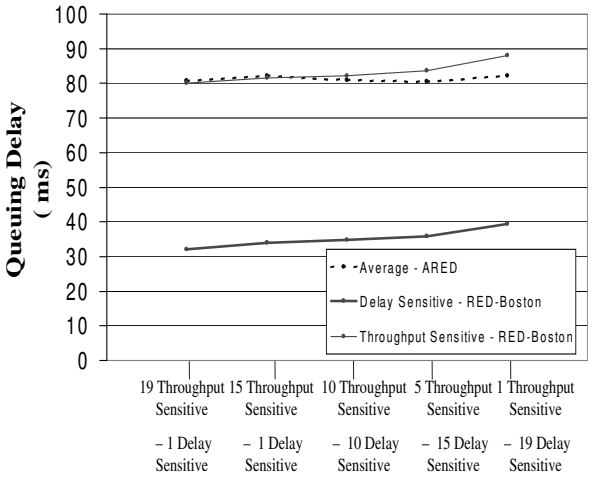

Figure 9: Simulation Set-1: Queuing Delays

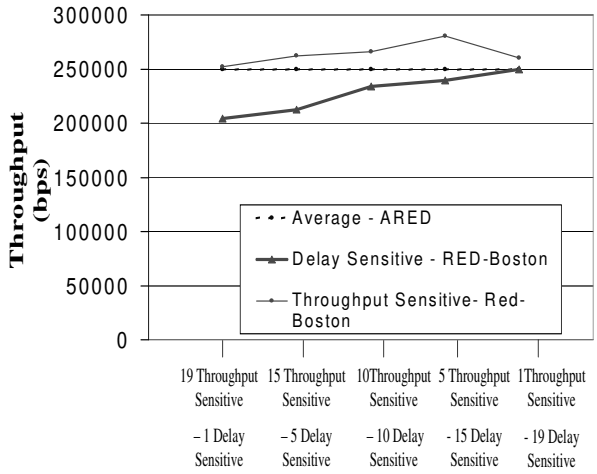

Figure 11: Simulation Set-1: Average Per Flow Throughput

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 9, it is clear that the aging component of the RED-Boston packet weight calculation prevents starvation of throughput-sensitive flows.

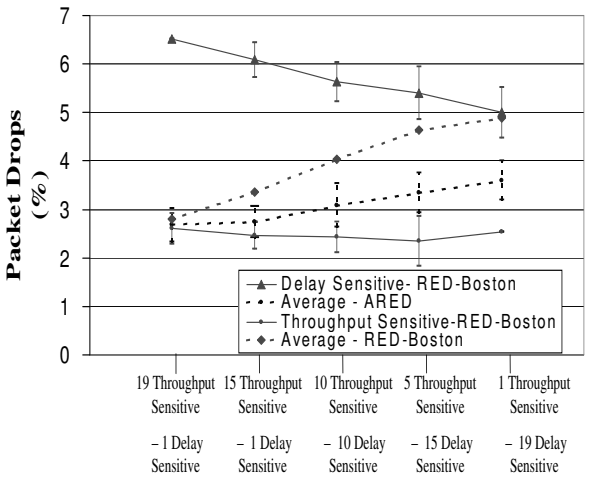

Figure 10: Simulation Set-1: Average Per Flow Percentage of Packets Dropped

Figure 10 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 QoS requirements. For RED-Boston, Figure 10 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, 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 number of delay-sensitive flows.

Figure 11 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 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 2.1, 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 12 presents the QoS for throughput-sensitive flows with a RED-Boston router normalized with respect to the QoS for throughput-sensitive flows traversing an ARED router. As described in Section 3, 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.

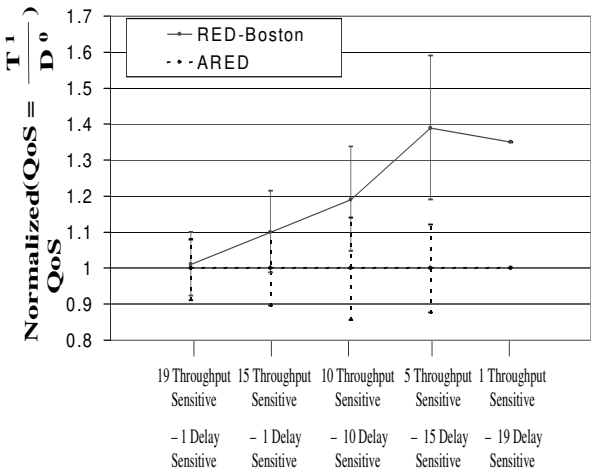

Figure 12: Simulation Set-1: Normalized QoS for Throughput-Sensitive Flows

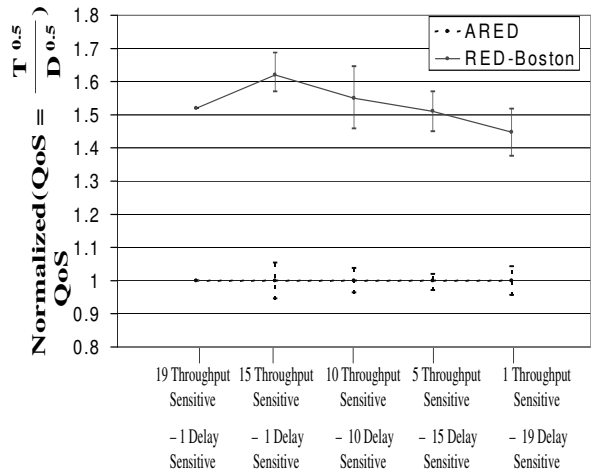

Figure 13: Simulation Set-1: Normalized QoS for Delay Sensitive Flows

Paralleling Figure 12, Figure 13 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, 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.

4.2 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 6. Delay hints for these TCP flows were uniformly varied from 16 ms to 112 ms in 16 ms increments. Figure 14 presents the details of the traffic mix for simulation set-2. This flow distribution was used for one simulation of a congested RED-Boston router which is compared against one 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 14.

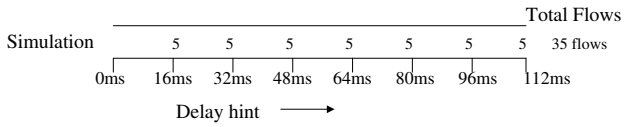

Figure 14: Simulation Set-2: Traffic Mix

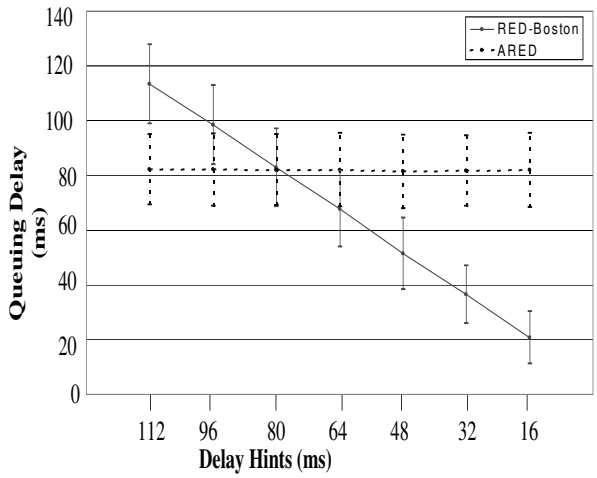

Figure 15: Simulation Set-2: Queuing Delays

Figure 15 shows the queuing delays in milliseconds for ARED and RED-Boston. While ARED provides essentially an 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 16 shows the differences in the percentage of packet drops for ARED and 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 [16] to counter act the higher loss rates.

Figure 17 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.

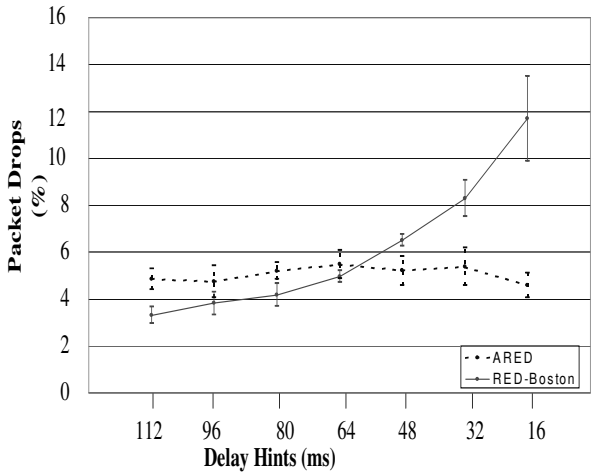

Figure 16: Simulation Set-2: Average Per Flow Percentage of Packets Dropped

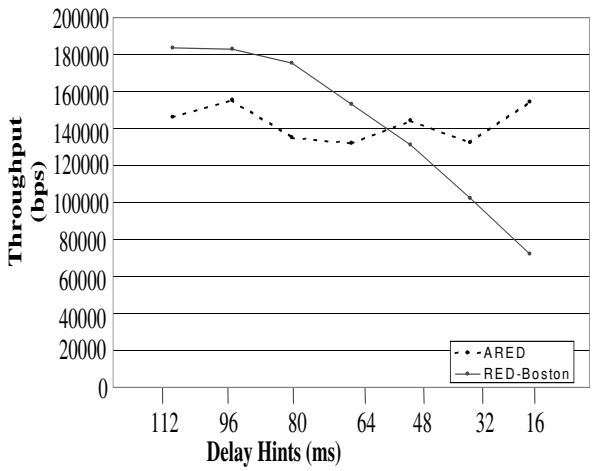

Figure 17: Simulation Set-2: Average Per Flow Throughput

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 1.

Figure 18 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.

5. SUMMARY

This paper presents RED-Boston, an active queue management technique designed to provide better QoS support to all applications under the current best-effort Internet environment. With RED-Boston, applications include a delay hint in their packets as an indication of their preference for either lower queuing delay or higher throughput at

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

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 1: Simulation Set-2: QoS Parameters

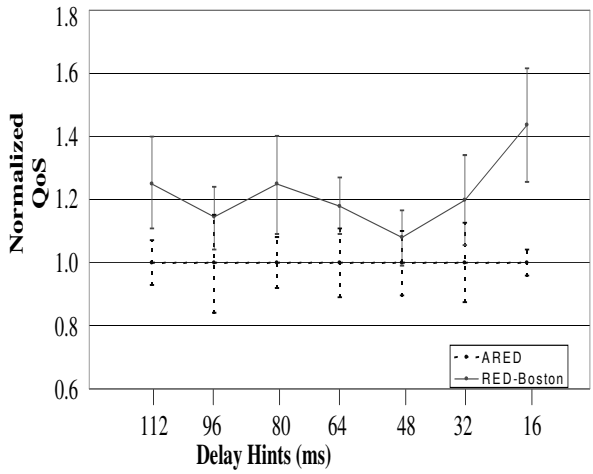

Figure 18: Simulation Set-2: Normalized QoS

the router. RED-Boston gives a relatively lower delay service to delay-sensitive flows and relatively higher throughput to throughput-sensitive flows. Thus, not only can low-bandwidth multimedia flows, such as interactive audio, obtain the low delays they need to support interactive communication, but delay-tolerant non-interactive flows, such as file downloading, can obtain the bandwidth they require to support efficient network transfer.

RED-Boston does not provide the delay and throughput guarantees provided by IntServ approaches, but nor does RED-Boston have the inherent scalability difficulties required by such per-flow reservations. RED-Boston does not pre-define classes of service as do DiffServ approaches, but instead provides a continuum of service classes. The service afforded by RED-Boston does not require additional monitoring or charging of either flows that take advantage of the low delay service or flows that take advantage of the high throughput service.

We have evaluated RED-Boston with varying percentages of delay-sensitive and throughput-sensitive flows and also with flows having varying delay and throughput requirements. We find that RED-Boston is able to adapt the overall router queue parameters to better suit the current traffic mix. Additionally, RED-Boston provides differentiated services by providing lower delay for the delay-sensitive flows and higher-throughput for throughput-sensitive flows. This improves QoS support for all flows in RED-Boston as compared to ARED.

Currently, RED-Boston expects flows to be responsive to

network congestion in a TCP-friendly fashion. Since RED-Boston provides a higher drop rate for flows with a low delay hint, 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 [19], so that the dropping probability can be calculated based on a delay hint and the flow rate, helping ensure bandwidth fairness for unresponsive traffic.

Another extension to RED-Boston could be interaction between cascaded RED-Boston routers. When dequeuing a packet, RED-Boston could be modified to update the cumulative amount of time the packet has waited in router queues, thus providing additional information for trading of delay and throughput for RED-Boston routers downstream.

6. REFERENCES

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

[2] J. Chung and M. Claypool. Dynamic-CBT and ChIPS - Router Support for Improved Multimedia Performance on the Internet. In *Proceedings of the ACM Multimedia Conference*, Nov. 2000.

[3] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, Feb. 1999.

[4] S. Floyd, R. Gummadi, and S. 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.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of ACM SIGCOMM Conference*, pages 45 – 58, 2000.

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

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

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

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

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

[11] Y. Liu and M. Claypool. Using Redundancy to Repair Video Damaged by Network Data Loss. In *Proceedings of IS&T/SPIE/ACM Multimedia Computing and Networking (MMCN)*, Jan.25-27 2000.

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

[13] 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*, Feb. 2000.

[14] 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.

[15] M. Parris, K. Jeffay, and F. Smith. Lightweight Active Router-Queue Management for Multimedia Networking. In *Proceedings of Multimedia Computing and Networking (MMCN), SPIE Proceedings Series*, Jan. 1999.

[16] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network Magazine*, Sep/Oct 1998.

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

[18] R. Rejaie, M. 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.

[19] I. Stoica, S. Shenker, and H. Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM Conference*, Sept. 1998.

[20] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of ACM SIGCOMM Conference*, Sept. 1999.

[21] 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.

[22] Y. Wang, M. Claypool, and Z. Zuo. An Empirical Study of RealVideo Performance Across the Internet. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

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