Dynamic-CBT – Better Performing Active Queue Management for Multimedia Networking

Jae Chung and Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
{goos|claypool}@cs.wpi.edu

Abstract

The explosive increase in the volume and variety of Internet traffic has placed a growing emphasis on congestion control and fairness in Internet routers. Approaches to the problem of congestion, such as active queue management schemes like Random Early Detection (RED) use congestion avoidance techniques and are successful with TCP flows. Approaches to the problem of fairness, such as Fair Random Early Drop (FRED), punish misbehaved, non-TCP flows. Unfortunately, these punishment mechanisms also result in a significant performance drop for multimedia flows that use flow control and do not scale since they require per-flow state information. We extend Class-Based Threshold (CBT) [4], and propose a new active queue management mechanism as an extension to RED called Dynamic Class-Based Threshold (D-CBT) to improve multimedia performance on the Internet. The performance of our proposed mechanisms is measured, analyzed and compared with other mechanisms (RED and CBT) in terms of throughput and fairness through simulation using NS. The study shows that D-CBT improves fairness among different classes of flows.

1. Introduction

The Internet has moved from a data communication network for a few privileged professions to an essential part of public life similar to the public telephone networks, while assuming the role of the underlying communication network for multimedia applications such as Internet phone, video conferencing and video on demand (VOD). As a consequence, the volume of traffic and the number of simultaneous active flows that an Internet router handles has increased dramatically, placing new emphasis on congestion control and traffic fairness. Complicating traditional congestion control is the presence of multimedia traffic that has strict timing constraints, specially *delay* constraints and variance in delay, or *jitter* constraints [1,2]. This paper presents a router queue management mechanism that addresses the problem of congestion and fairness, and improves multimedia performance on the Internet. Figure 1 shows some of the current and the proposed router queue mechanisms.

There have been two major approaches suggested to handle congestion by means other than traditional drop-tail FIFO queuing. The first approach uses packet or link scheduling on multiple logical or physical queues to explicitly reserve and allocate output bandwidth to each class of traffic, where a class can be a single flow or a group of similar flows. This is the basic idea of various Fair Queuing (FQ) disciplines and the Class-Based Queuing (CBQ) algorithm [3]. When coupled with admission control, the mechanism not only suggests a solution to the problem of congestion but also offers potential performance guarantees for the multimedia traffic class. However, this explicit resource reservation approach would change the "best effort" nature of the current Internet, and the fairness definition of the traditional Internet may no longer be preserved. Adopting this mechanism would require a change in the network management and billing practices. Also, the algorithmic complexity and state requirements of scheduling make its deployment difficult [4].

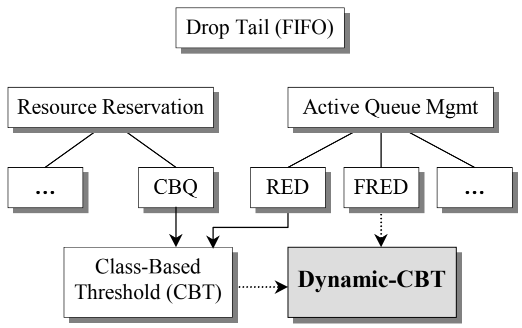


Figure 1: Router Queue Mechanisms (shaded is proposed)

The second approach, called Active Queue Management, uses advanced packet queuing disciplines other than traditional FIFO drop-tail queuing on an outbound queue of a router to actively handle (or avoid) congestion with the help of cooperative traffic sources. In the Internet, TCP recognizes packet loss as an indicator of network congestion, and its back-off algorithm reduces transmission load when network congestion is detected [5]. One of the earliest and well-known active queue management mechanism is Random Early Detection (RED), which prevents congestion through monitoring outbound buffers to detect impending congestion, and randomly chooses and notifies senders of network congestion so that they can reduce their transmission rate [6]. While fairly handling congestion for TCP flows, RED has a potentially critical problem that non-TCP flows that are unresponsive or have greedier flow-control mechanisms than TCP can take more share of the output bandwidth than TCP flows [4,7]. In the worst case, it is possible for non-TCP flows, especially unresponsive ones, to monopolize the output bandwidth while TCP connections are forced to transmit at their minimum rates. This unfairness occurs because non-TCP flows reduce transmission load relatively less than TCP flows or do not reduce at all, and the same drop rate is applied to every flow.

This unfairness could be a serious problem in a near future as the number of Internet multimedia flows increases. Delay sensitive multimedia applications typically use UDP rather than TCP because they require in-time packet delivery and can tolerate some loss, rather than the guaranteed packet delivery with potentially large end-to-end delay that TCP produces. Also, they prefer the periodic packet transmission characteristics of UDP rather than the bursty packet transmission characteristics of TCP that can introduce higher receiver side jitter. Multimedia UDP applications either do not use any flow-control mechanism or use their own application-level flow control mechanisms that are rate-based rather than window based and hence tend to be greedier than that of TCP taking the multimedia Quality of Service (QoS) requirements into account [10].

In addressing the problem of fairness, there have been strong arguments that unresponsive or misbehaving flows should be penalized to protect well-behaved TCP flows[1] [8]. Fair Random Early Drop (FRED) is an active queue management approach that incorporates this argument [7]. FRED adds per-active-flow accounting to RED, isolating each flow from the effects of others. It enforces fairness in terms of output buffer space by strictly penalizing unresponsive or misbehaving flows to have an equal fair share while assuring packets from flows that do not consume their fair share are transmitted without loss. FRED achieves its purpose not only in protecting TCP flows from unresponsive and misbehaving flows but also in protecting fragile TCP connections from robust TCP connections. However, per-active-flow accounting is expensive and might not scale well. FRED also has a potential problem that its TCP favored per-flow punishment could unnecessarily discourage flow-controlled interactive multimedia flows. Under FRED, incoming packets for a well-behaved TCP flow consuming more than their fair share are randomly dropped applying RED's drop rate. However, once a flow, although flow-controlled, is marked as a non-TCP friendly flow, it is regarded as an unresponsive flow and all incoming packets of the flow are dropped when it is using more than its fair share. As a result, a flow-controlled multimedia UDP flow, which may have a higher chance to be marked, will experience more packet loss than a TCP flow and be forced to have less than its fair share of bandwidth.

Parris *et al.* [4] propose a new active queue management scheme called Class-Based Threshold (CBT), which releases UDP flows from strict per-flow punishment while protecting TCP flows by adding a simple class-based static bandwidth reservation mechanism to RED. In fact, CBT implements an explicit resource reservation feature of CBQ on a single queue that is fully or partially managed by RED without using packet scheduling. Instead, it uses class thresholds that determine ratios between the number of queue elements that each class may use during congestion. CBT defines three classes: tagged (multimedia) UDP[2], untagged (other) UDP and TCP. For each of the two UDP classes, CBT assigns a pre-determined static threshold and maintains a weighted-average number of enqueued packets that belong to the class, and drops the incoming class' packets when the class average exceeds the class threshold. By applying a threshold test to each UDP class, CBT protects TCP flows from unresponsive or misbehaving UDP flows, and also protects multimedia UDP flows from the effects of other UDP flows. CBT avoids congestion as well as RED, has less overhead and improves multimedia throughput and packet drop rates compared to FRED. However, as in the case of CBQ, the static resource reservation mechanism of CBT could result in poor performance for rapidly changing traffic mixes and is arguably unfair since it changes the best effort nature of the Internet.

To eliminate the limitations due to the explicit resource reservation of CBT while preserving its good features from class-based isolation, we propose *Dynamic-CBT (D-CBT)*. D-CBT fairly allocates the bandwidth of a congested link to the traffic classes by dynamically assigning the UDP thresholds such that the sum of the fair share of flows in each class is assigned to the class at any given time era if the mix of flows changes.

We use an event driven network simulator called NS (version 2) [9] to evaluate D-CBT. We implement CBT in NS, extend it to D-CBT, and compare the performance of D-CBT with that of RED and CBT. In the evaluation, our primary focus is on the effect of heterogeneously flow-controlled traffic on the behavior of the queue management mechanisms especially on fairness.

Section 2 discusses CBT and Section 3 presents D-CBT in detail. Section 4 presents our validation of CBT in NS and section 5 describes the flow-controlled multimedia traffic generator that we used for tagged flows. Section 6 describes our simulation setup, Section 7 analyzes and evaluates D-CBT and Section 8 concludes our research.

2. Class-Based Threshold (CBT)

Before describing D-CBT, we discuss the design of Class-Based Threshold (CBT) [4] which D-CBT extends. As discussed briefly in Section 1, the main idea behind the design of CBT is to apply class-based isolation on a single queue that is fully or partially managed by RED without using packet scheduling. Instead of using packet scheduling on multiple logical queues, CBT regulates congestion-time output bandwidth for *n* classes of flows using a RED queue management mechanism and a threshold for each of the *n-1* classes of flows, which is the average number of queue units that a class may use.

CBT categorizes flows into three classes, which are TCP, tagged (multimedia) UDP and untagged (other) UDP, and assigns a pre-determined static threshold for each of the two UDP classes, assuming that UDP flows are mostly unresponsive or misbehaving and need to be regulated. When a UDP packet arrives, the weighted-average for the appropriate class is updated and compared against the threshold for the class to decide whether to drop the packet before passing it to the RED algorithm. For the TCP class, CBT does not apply a threshold test but directly passes incoming packets to the RED test unit. This is the first design of CBT, called "CBT with RED for all". In the second design, called "CBT with RED for TCP", only TCP packets are subjected to RED's early drop test, and UDP packets that survive a threshold test are directly enqueued to the outbound queue that is managed by RED. Another difference from the first design is that RED's average queue size is calculated using only the number of enqueued TCP packets. CBT with RED for TCP is based on the assumption that tagged (multimedia) UDP flows as well as untagged (other) UDP flows are mostly unresponsive, and it is of

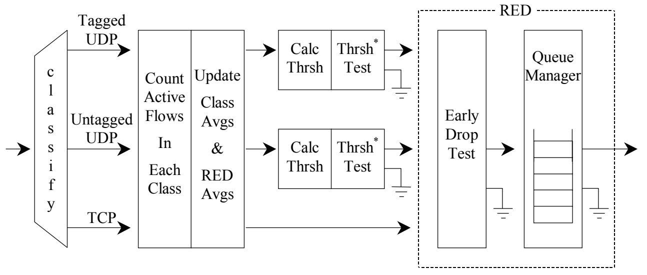
[1] A *well-behaved flow* (or TCP friendly) is defined as a flow that behaves like a TCP flow with a correct congestion avoidance implementation. A flow-controlled flow that acts different (or greedier) than well-behaved flow is a *misbehaving flow*.

[2] Tagged (multimedia) UDP flows can be distinguished from other (untagged) UDP flows by setting an unused bit of the Type of Service field in the IP header (Version 4).

no use to notify these traffic sources of congestion earlier. D-CBT is extended from CBT with RED for all. In the rest of this paper, CBT refers to CBT with RED for all.

3. Dynamic-CBT (D-CBT)

D-CBT enforces fairness among classes of flows, and gives UDP classes better queuing resource utilization. Figure 2 shows the design of D-CBT. The key difference from CBT is (1) the dynamically moving fair thresholds and (2) the UDP class threshold test that actively monitors and responds to RED indicated congestion. To be more specific, by dynamically assigning the UDP thresholds such that the sum of the fair average queue resource share of flows in each class[3] is assigned to the class at any given time, D-CBT fairly allocates the bandwidth of a congested link to the traffic classes. Also, the threshold test units, which are activated when RED declares impending congestion (i.e. *red_avg* > *red_min*), coupled with the fair class thresholds, allow the UDP classes to use the available queue resources more effectively than in CBT, in which each UDP class uses the queue elements an average of no more than its fixed threshold at any time. Looking at it from a different view, D-CBT can be thought of a Class-Based FRED-like mechanism that does *per-class-accounting* on the three classes of flows.



* Threshold Test is activated when *red_avg* > *red_min*

Figure 2: Design of Dynamic-CBT (D-CBT)

As in CBT, D-CBT categorizes flows into TCP, tagged UDP and untagged UDP classes. However, unlike the class categorization of CBT in which flow-controlled multimedia flows are not distinguished from unresponsive multimedia flows (all tagged), D-CBT classifies UDP flows into flow-controlled multimedia (tagged) UDP and other (untagged) UDP. Thus, only flow-controlled (or responsive) multimedia flows are categorized into tagged UDP and all other flows including unresponsive multimedia flows are classified into untagged UDP. The objective behind this classification is to protect flow-controlled multimedia flows from unresponsive multimedia flows, and encourage multimedia applications to use congestion avoidance mechanisms, as does TCP. We believe that there are advantages in categorizing UDP traffic in this way for the following reasons: first, multimedia applications are the primary flows that use high bandwidth UDP; second, by categorizing flows by their congestion responsiveness characteristic (i.e. TCP friendly, flow-controlled but misbehaving multimedia and unresponsive flows), different management can be applied to the classes of differently flow-controlled flows.

In fact, in determining the fair UDP thresholds, D-CBT calculates the fair average output buffer share of the tagged UDP class from the average queue length that is maintained by RED, and that of the untagged UDP class from the RED's minimum threshold (plus a small allowance). This is based on the assumption that tagged flows (or flow-controlled multimedia) can respond to network congestion and will actively try to lower the average length of a congested queue on notification of congestion. Therefore, they are allowed to use the impending congestion state queue buffers (i.e. *red_avg – red_min* when *red_avg* > *red_min*) up to their fair share of the average. However, unresponsive (untagged) flows, which have no ability to respond to network congestion, are not allowed to use the impending state queue buffers at impending congestion. Actually, we allow the unresponsive UDP class to use a small fraction of the impending state queue buffers, which is 10% of (*red_max – red_min*) * *untagged_UDP_share* (when the maximum early drop rate is 0.1), to compensate for the effect of needless additional early drops for the class. Figure 3 shows the tagged and untagged UDP threshold calculations.

$$Tagged\ \ UDP\ Thsh = \frac{num\ \ of\ \ tagged\ \ active\ \ flows}{total\ \ num\ \ of\ \ active\ \ flows} \times red\ \ avg$$

$$Unagged\ \ UDP\ Thsh = \frac{num\ \ of\ \ untagged\ \ active\ \ flows}{total\ \ num\ \ of\ \ active\ \ flows} \times \left[red\ \ min + \left(red\ \ max - red\ \ min\right) \times 0.1\right]$$

Figure 3: Tagged and Untagged UDP Threshold Calculations

In the design of D-CBT, the existence of the active flow counting unit is a big structural difference from CBT. In order to calculate a fair threshold (or average queue resource share) for each class, D-CBT needs class state information, and therefore keeps track of the number of active flows in each class. Generally, as in FRED, active flows are defined as ones whose packets are in the outbound queue [7]. However, we took slightly different approach in detecting active flows, in that an active flow is one whose packet has entered the outbound queue unit during a certain predefined interval since the last time checked. In D-CBT, an active flow counting unit that comes right after the classifier maintains a sorted linked list, which contains a flow descriptor and its last packet reception time, and a flow counter for each class. Currently, the flow descriptor consists of a destination IP address and the flow ID (IPv6). However, assuming IPv4, this could be replaced by source and destination address, although this would redefine a flow as per source-destination pair. Tagged UDP flows can be identified in IP layer using the protocol field and an unused bit in the type-of-service field for IPv4. For Ipv6, this can be done using the next-header field and the priority field.

For an incoming packet after the classification, the counting unit updates an appropriate data structure by inserting or updating the flow information and the current local time. When inserting new flow information, the flow counter of the class is also increased by one. The counting unit, at a given interval (set to 300ms in our implementation), traverses each class' linked list, deletes the old flows and decreases the flow counter. The objective behind this probabilistic active flow counting approach is twofold: (1) D-CBT does not necessarily require an exact count of active flows as do

[3] Fair class shares are calculated based on the ratio between the number of active flows in each class over the total number of flows in all classes.

other queue mechanisms that are based on flow-based-accounting, although a more exact count is better for exercising fairness among flow classes; and (2) it might be possible to improve the mechanism's packet processing delay by localizing the counting unit with the help of router's operating system and/or device. For example, the traversing delete is a garbage collection-like operation that could be performed during the router's idle time or possibly processed by a dedicated processor in a multiprocessor environment. In our simulator implementation, we used a sorted linked-list data structure that has the inserting and updating complexity of *O(n)*, and the traversing complexity of *O(n)*, where *n* is the number of flows of a class. Assuming that a simple hash table is used instead, the complexity of inserting and updating drops to *O(1)*, while the complexity of the traverse delete will remain *O(n)*.

When an incoming packet is updated or inserted, according to its flow identification, into its class data structure at the counting unit, D-CBT updates the RED queue average, the tagged UDP average and the untagged UDP average, and passes the packet to an appropriate test unit as shown in Figure 2. Note that for every incoming packet all of the averages are updated using the same weight. This is to apply the same updating ratio to the weighted-averages, so that a snapshot in time at any state gives the correct average usage ratio among the classes. Using the three averages and the active flow count for each class, the UDP threshold test units calculate the fair thresholds for the tagged and untagged UDP classes, and apply the threshold test to incoming packets of the class when the RED queue indicates impending congestion. UDP packets that survive an appropriate threshold test are passed to the RED unit along with the TCP flows as in CBT.

Thus, D-CBT is designed to provide traditional fairness between flows of different characteristics by classifying and applying different enqueue policies to them, and restrict each UDP class to use the queue buffer space up to their average bandwidth share. We hypothesize that the advantages of D-CBT are the following: first, D-CBT avoids congestion as well as RED with the help of responsive traffic sources; and second, assuming that the flows in a class (especially the tagged UDP flows) use flow control mechanisms with similar congestion responsiveness characteristics, D-CBT will fairly assign bandwidth to each flow with much less overhead than FRED, which requires per-flow state information. Even if the tagged flows do not use their fair share, D-CBT will still successfully assign bandwidth fairly to each class of flows, protecting TCP from the effect of misbehaving and unresponsive flows and also protecting the misbehaving (flow-controlled multimedia) flows from the effect of unresponsive flows. Lastly, D-CBT gives tagged (flow-controlled multimedia) flows a better chance to fairly consume the output bandwidth than under FRED by performing per-class punishments instead of the strict per-flow punishment.

4. CBT Validation

Since D-CBT extends CBT, we first implemented CBT in NS. We took the RED implementation in NS and extended it to CBT (with RED for all) [4]. After the implementation, we validated the RED and the CBT implementations by simulating the experiments in [4] and comparing the RED and CBT results with the simulated ones. Figure 4 shows the experimental network consisted of two switched 100 Mbps Ethernet LANs (source and destination LAN) that were interconnected by a full-duplex 10Mbps Ethernet. For traffic sources, they used 6 video ProShare multimedia applications that generate approximately 210 Kbps of traffic each, 240 FTP-TCP bulk transfers and 1 UDP non-multimedia application that sent 1 Kbyte packets at 10 Mbps (the maximum capacity of the network), of which the scheduling is shown in Figure 5. To simulate a large RTT for the TCP connections, the kernel on each machine was modified to introduce a delay in transmitting packets from each connection. Refer to [4] for the further experiment setup details.

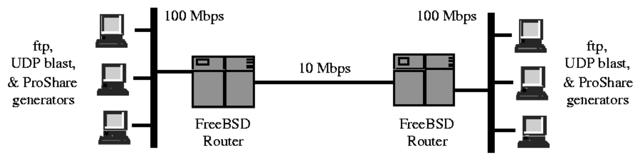

100 Mbps
100 Mbps
ftp, UDP blast, & ProShare generators
10 Mbps
FreeBSD Router
FreeBSD Router
ftp, UDP blast, & ProShare generators

Figure 4: Experimental Network in [4]

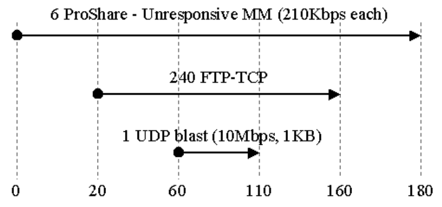
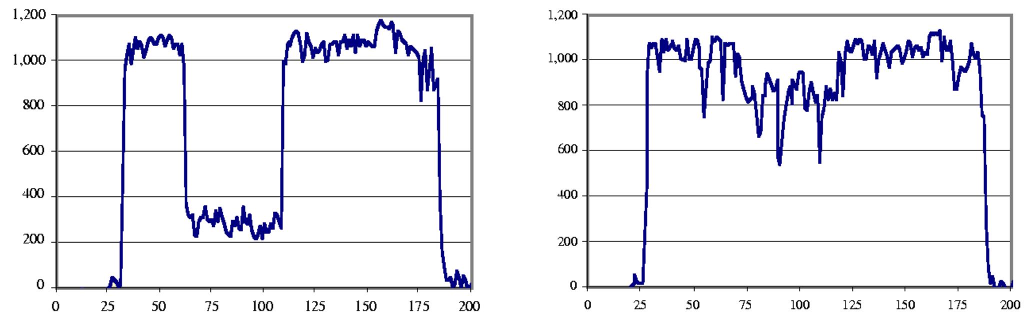


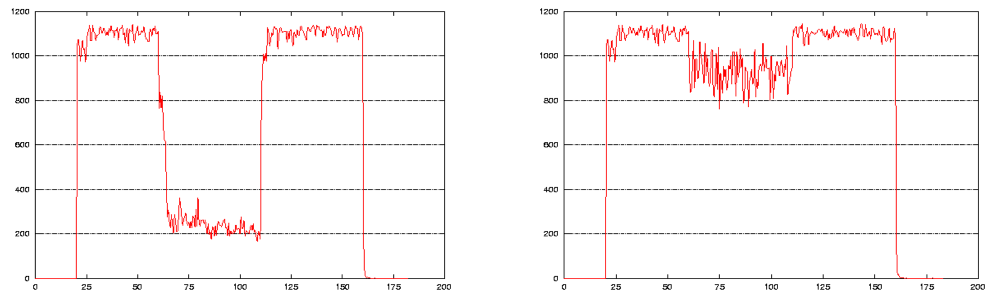
Figure 5: Traffic Source Schedule

Similarly, our simulation used the network topology of 100 Mbps source and destination Ethernet LANs, each of which consisted of 248 nodes (147 hosts and a router), connected by a 10 Mbps link. However, instead of introducing a large delay in the nodes, we give the 10Mbps link the delay of 20ms. For the traffic source, we assumed many of the parameter settings, since no detailed source behavior or parameter settings are described in [4]. To simulate the ProShare multimedia application, we used a CBR traffic generator that transmitted 6 Kbytes frames at the rate of 210 Kbps, each of which was broken into six 1 Kbyte tagged packets at the underlying UDP layer. For the TCP (Reno) connection, we set the maximum packet size to 300 Bytes and the maximum congestion window size to 10 packets.

Each router is set to have a 60 packet long output queue. For RED, the maximum and minimum thresholds were 30 packets and 15 packets, the weight was 0.002 for the weighted queue average calculation, and the maximum early drop probability was 0.1. For CBT, in addition to the RED settings, the tagged UDP class threshold (mm-th) was set to 10 packets, and the untagged UDP class threshold (udp-th) was set to 2 packets. These class thresholds were determined based on an assumption that the average of the average queue length would be about 26 packets throughout the experiments (or the simulation), and all packets are of the same size. By assigning the maximum average of a 10-packet space of the queue to the tagged flow class, it approximately gets the maximum bandwidth of 3.8 Mbps, 10/26 of the network link bandwidth, at congestion. This was enough for the 6 ProShare flows whose aggregated average bandwidth requirement was approximately 1.3 Mbps. Similarly, for the untagged UDP flows, by assigning the average of a 2-packet space the class for congestion, approximately 0.8 Mbps of output was reserved for the class. From the above calculation, the aggregated


RED CBT with RED for all

(a) Experimental Results in [4]


RED CBT with RED for all

(b) Simulated Results

Figure 6: Aggregate TCP Throughput (X-axis = Seconds, Y-axis = Kbytes/Sec)

throughput of TCP, which should have been about 8.7 Mbps (or 1088 Kbytes/Sec) when TCP flows are sharing the bandwidth with the ProShare flows, should have dropped down to approximately 7.9 Mbps (or 988 Kbytes/Sec) when the UDP blast joins. However, TCP flows got a little bit less bandwidth than the calculation due to their smaller packet size. By using smaller packets, TCP agents would operate in a larger congestion window size and transmit more packets into the network resulting in more TCP packet drops at the router. However, since the TCP bandwidth loss due to a packet drop was reduced, the TCP bandwidth loss was not significant.

Figure 6 shows the aggregated TCP throughput (in Kbytes/Sec) under RED and CBT for (a) the experiment presented in [4] and (b) the simulation we conducted (the throughput graph for the experiments is copied directly from the paper). By comparing the experimental result and the simulated result under RED, one can see that the traffic source parameter settings we deduced for the unspecified ones are similar to the settings used in the experiments. Also, the results imply that the TCP and RED implementation on NS are comparable to the ones used in the experiment. Next, the aggregated TCP throughputs under CBT for the experiment and the simulation show that our NS implementation is correct, although the experimental result shows several sharply dropping aggregated TCP throughputs during the period when the UDP blast joins. We believe that this phenomenon is caused by experimental "noises" from the real world.

Comparing the experimental and the simulated results carefully, one can notice the delayed throughput of the experiment at the beginning and at the end. This could be simply from the fact that [4] used a transmission source scheduling that is slightly different than shown in Figure 5. Assuming that the same schedule is used, it is possible that this difference comes from the mechanism in the host kernels that introduces a delay for TCP packet transmissions could be set to have a higher delay than that of the network link in the simulation. Another possible factor is the routing delays of the routers in the experiment. In fact, the router implementation in NS does not have a routing delay and thus does not have an inbound queue. It appears that each router in the experiment is set to have a large inbound queue. In this situation, when a severe overload occurs in the inbound link that is considerably off the router's switching capacity (or processing delay), the incoming packets wait a significant time in the inbound queue before being processed or even dropped. At the period when TCPs start transmission the overload occurs due to the slow start algorithm with the relatively large congestion window sizes. Likewise, it seems that when the UDP blast starts, a very significant overload occurred at the inbound link. These effects, we believe, contributed to the delayed transmission at the router, and possibly the sharply dropping aggregated TCP throughput during the period of 60 to 110 seconds.

This section showed the validation of the TCP, RED and the newly added CBT implementations in NS. By simulating the [4] experiment and comparing the results with the experimental results, we showed that the TCP, RED, and the CBT implementations are comparable to the ones used in the experiments. The next section briefly describes the behavior of flow-controlled multimedia traffic source that we designed and used as tagged traffic sources for the simulation that compares RED, CBT and D-CBT.

5. Flow-Controlled Multimedia

At the time we started this work, NS did not support flow-controlled multimedia traffic. We designed and implemented a multimedia traffic generator that responds to network congestion, extended from media scaling techniques proposed in [12]. The traffic generator, MM_APP, reduces or increases transmission rate by decreasing or increasing the transmission interval with a fixed frame size. The goal is to use this framework to achieve an Additive Increase Multiplicative Decrease (AIMD) algorithm. TCP uses AIMD congestion control and is required to avoid congestion collapse on the Internet [8].

Scale Value	Media Encoding and Transmission Policy Set	Avg. Transmission Rate (Kbps)
4	A	1100
3	B	900
2	C	700
2	D	500
1	E	300

Table 1: Example Media Scale Assignment

The MM_APP applications use sender and receiver behavior. Before transmitting the actual data, the sender and the receiver agree on five scale values (0 to 4), each of which is assigned to a different media encoding method and transmission policy (i.e. which frame to transmit) pair. The scale value 0 is assigned to a set from which a predetermined minimum sustainable media quality can be achieved, the next value is assigned to sets from which a better media quality can be achieved, and so on. It is assumed that the media encoding and transmission policy sets are carefully chosen so that the transmission rates resulting from the sets increase linearly as the scale value increases. Table 1 shows an example assignment.

Thus, the sender has five discrete and linearly increasing transmission rates assigned to the scale values, and starts from scale 0 transmitting at the lowest rate. The receiver detects congestion (or lack of it), determines the next transmission rate of the sender in terms of scale value, and notifies the sender of this scale value. The sender, being notified of the scale value, simply changes the transmission rate by using media encoding and transmission policy assigned to the scale value.

In detecting congestion, the receiver uses frame loss as the network congestion indicator. There are two circumstances where the receiver claims frame loss. The first is when the receiver gets a frame whose sequence number is greater than the expected sequence number. The second is when the receiver does not receive any frames within a timeout interval. Proper setting of the timeout interval is critical. A timeout interval that is set too short will claim false frame losses, which will make the sender reduce the transmission rate needlessly. On the other hand, a timeout interval that is set too long will fail to detect multiple sequential frame loss effectively such that the sender reduces transmission rate later than other competing connections, which could result in an unfair portion of bandwidth. In our implementation, the Round Trip Time (RTT) of the connection is greater than the longest possible frame transmission interval:

```
RTT > Max_Intrvl: TOI = RTT
RTT ≤ Max_Intrvl: TOI = Max_Intrvl + α
(where 0 < α < RTT)
```

The receiver, when detecting congestion, reduces its scale value to half (integer division) and notifies the sender of this value by sending a small packet. When the receiver detects no network congestion within a RTT from the last checkpoint, it increases the scale value by one and notifies the sender of this value. This design of drop scale to half at congestion, and increase one scale up at a RTT is motivated by the fast recovery algorithm that is found in Reno TCP implementations [13] and the TCP-friendly definition [8].

As briefly mentioned earlier, MM_APP directly associates transmission rates to scale values without targeting a specific media encoding and transmission policy. It assumes that every media encoding and transmission policy pair associated to the scale values generates traffic with a fixed frame size. In other words, it assumes that the transmission intervals are the only factors that cause the rate changes. Therefore, the resulting traffic can be characterized by CBR traffic of a fixed frame size and various transmission intervals associated with the scale values. Although MM_APP is not tied to a specific multimedia application, it is useful in that it is easy to change the transmission rate associated to each scale value and then test the media scaling scheme, thus eliminating the effect of a specific traffic characteristic for a particular application.

We also tested our multimedia flow-control scheme with a specific video application that is based on the MPEG-1 encoding scheme [14]. MPEG_APP implements five sets of MPEG-1 encoding and transmission policies and associates them with scale vales - in fact, it only changes the frame transmission policy leaving the encoding scheme unchanged. MPEG-1 encodes video at a given frame rate and picture quality, generating a stream of frame types I, P and B, associated with a typical Group of Pictures (GOP), such as IBBPBBPBB. Among the three frame types, only I-frames can be decoded on their own. The decoding of a B-frame relies on a pair of I-frames and/or P-frames that come before and after the B-frame and the decoding of a P-frame relies on an I-frame or P-frame that comes before the P-frame. MPEG_APP supports MPEG-1 streams using the common GOP patterns IBBPBBPBB and IBBPBBPBBPBB. Figure 7(b) shows the transmission policies on the latter stream patterns, which is carefully selected keeping the dependencies in mind [15].

MPEG_APP reads in an MPEG-1 trace file that contains frame information for a stream with the maximum frame rate (scale 4) as input along with the maximum frame rate and the longest possible frame transmission interval that is used for congestion detection at the receiver side. At every scale transmission interval, MPEG_APP reads the frame information from the input file, and determines whether or not to transmit this frame using the current scale value and the transmission policy associated with the scale value.

Figure 7(a) shows an example input file that contains frame information for a 30 frames per second IBBPBBPBB pattern stream in which the sizes of the I-, P- and B-frames are 11 KB, 8 KB and 2 KB, respectively. The frame sizes used in this example are the mean frame size of each type obtained while playing a short high quality MPEG-1 news clip. Figure 7(b) shows each transmission policy assigned to scale values with the estimated average transmission rate for the input trace file. Figure 7(c) visualizes the estimated transmission rate in Figure 7(b). The almost linearly growing estimated average transmission rates shows that the assignment of the transmission policies to the scale values works well with the given example MPEG-1 stream. This is because the linear increment of scale results in linear increment of transmission rate and the exponential decrement of scale results in exponential decrement of transmission rate.

```
I        11000        P        8000
B        2000         B        2000
B        2000         B        2000
P        8000         I        11000
B        2000         .        .
B        2000         .        .
```

Figure 7(a): Input Trace File (in bytes)

Maximum Frame Rate (Scale 4) = 30 frame/sec		
Scale	Transmission Policy (Pattern 1)	Estimated Average Trans. Rate (Kbps)
4	I B B P B B P B B I	1056
3	I B P B P B I	896
2	I P P I	736
1	I P I	544
0	I I	352

Figure 7(b): Transmission Policies and Associated Rates

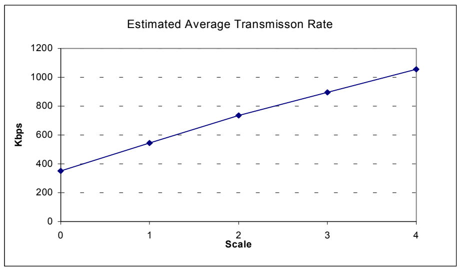


Figure 7(c): Estimated Average Transmission Rate of the Example in Figure 7(b).

We ran a series of simulations to validate the NS implementation of MM_APP and MPEG_APP and to measure their fairness when competing for bandwidth with TCP flows under RED queue management. Here, we present and compare the results of two simulations of which one has no flow control mechanism for the multimedia traffic sources and the other has the flow control mechanism for the multimedia traffic as just described. Figure 8 shows the network topology and the application flows used for the simulation.

Each link that connected a source or destination node and a network node had 10 Mbps link capacity and 5ms of delay. The link that connected the two network nodes has 6 Mbps of link capacity and 20ms of delay. The network node n1 used RED queue management, for which the parameter set used was chosen from one of the sets that are recommended by [6]. For traffic sources, 6 FTP, 2 MM_APP and 2 MPEG_APP traffic generators were used, where FTP used TCP Reno and the others used UDP as the underlying transport agent. All the TCP agents were set to have a maximum congestion window size of 20 packets and maximum packet size of 1500 bytes. The UDP agents were also set to have a maximum packet size of 1500 bytes. The MM_APP traffic generators used the transmission rates shown in Table 1, that is 300, 500, 700, 900, 1100 Kbps, for scale 0 to 4 transmission rates, respectively. The MPEG_APP traffic generators used the transmission policies and the trace file shown in Figure 7, which generated traffic rates of 352 Kbps to 1056 Kbps. The MM_APP and MPEG_APP were run without flow control for the first simulation transmitting at its maximum rate (i.e. 1100 Kbps for MM_APP and 1056 Kbps for MPEG_APP) and with flow control on for the second simulation.

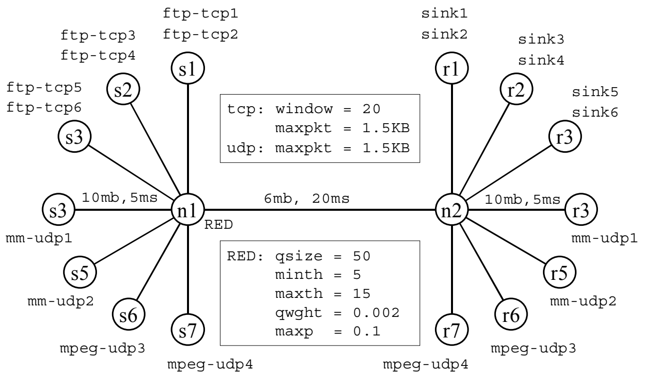
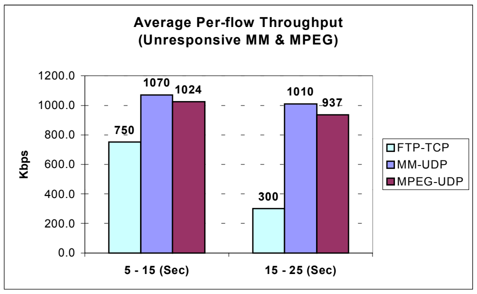
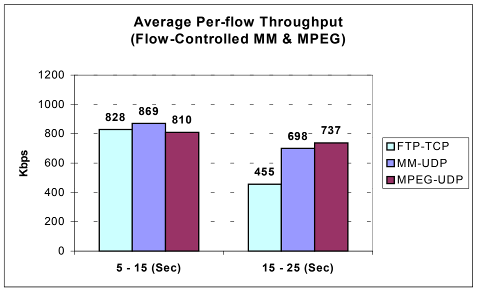


Figure 8: MM Test Network Topology and Flows

Both the simulations started with five FTP applications, 1 MM_APP and 1 MPEG_APP, and after 15 seconds the remaining traffic sources joined. For the first 15 seconds, the available bandwidth share for each connection was about 857 Kbps, and for the next 10 seconds, the share went down to 600 Kbps. For the throughput measurements, we omitted the first 5 seconds to eliminate the startup effect of unstable TCP and RED behaviors on the fairness at the initial stage.



(a) Unresponsive Multimedia



(b) Flow-Controlled Multimedia

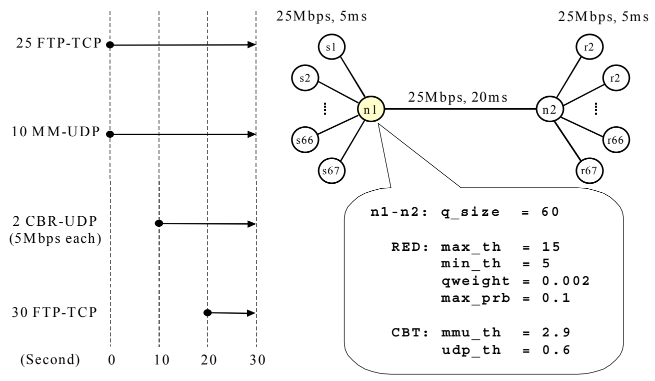
Figure 9: Average Per-Flow Throughput for MM Test

Figure 9 compares the average per-flow throughput of FTP, MM_APP and MPEG_APP for the two cases. Figure 9 (a) shows that unresponsive multimedia flows get the bandwidth close up to their needs while FTP over TCP back off and share the leftover bandwidth. However, as shown in Figure 9 (b), the responsive multimedia flows share network bandwidth fairly with TCP flows. Even during the second part of the simulation, seconds 15-25, the multimedia streams do receive more their fair share of bandwidth when bandwidth becomes even more scarce, but still do not completely starve out the TCP flows.

We briefly presented the behavior and the responsiveness of our rate-based multimedia flow control mechanism, which is compatible to that of TCP while it can be greedier than TCP under high network congestion. In the following series of simulations MM_APP traffic generator (flow-controlled) is used as tagged flow sources. Refer to [10] for more details on the responsive multimedia traffic generators.

6. Simulation: RED vs. CBT vs. D-CBT

We ran an NS simulation for each of RED, CBT and D-CBT to compare the effect of the router queue management mechanisms on fairness in network bandwidth allocation. Every simulation had the exactly same settings except for the network routers, each of which was set to use one of the above three outbound queue management mechanisms. The network topology and the traffic source schedules are shown in Figure 10.



```
n1-n2: q_size   = 60

  RED: max_th   = 15
       min_th   = 5
       qweight  = 0.002
       max_prb  = 0.1

  CBT: mmu_th   = 2.9
       udp_th   = 0.6
```

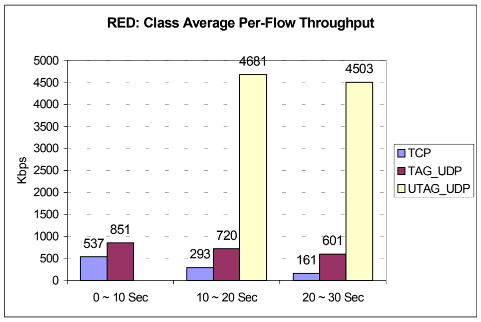
Figure 10: Simulation Scenario and Network Setup

For traffic sources, 55 FTP, 10 flow-controlled multimedia traffic generators, MM_APP (tagged) and 2 CBR (untagged) traffic generators were used, where FTP used TCP Reno and the others used UDP as the underlying transport agent. The network packet size was set to 1Kbyte. The MM_APP traffic generators, which react to congestion using 5 discrete media scales with a "cut scale by half at frame loss, up scale by one at RTT" flow control mechanism, used 300, 500, 700, 900 and 1,100Kbps for scale 0 to 4 transmission rates, with a fixed frame size of 1Kbyte. The CBR sources were set to generate 1Kbyte packets at a rate of 5Mbps.

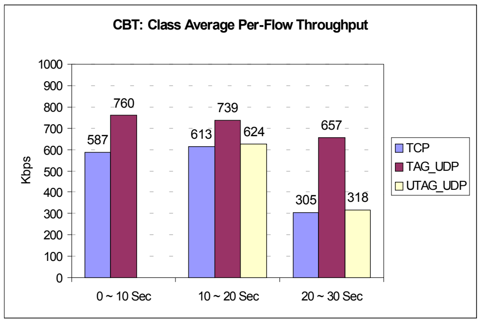
Network routers were assigned a 60-packet long physical outbound queue. The RED parameters, which are shown in Figure 10, were chosen from one of the sets that are recommended by Floyd and Jacobson [6]. For CBT, besides the RED parameters, the tagged and untagged class thresholds (denoted as *mmu_th* and *udp_th* in the figure) were set to 2.9 packets and 0.6 packets to force each UDP flow to get about their fair bandwidth shares during 0 to 20 seconds. D-CBT also shares the RED settings, but since each threshold is assigned dynamically to the fair share of each class, no threshold setup was necessary.

7. Result and Analysis

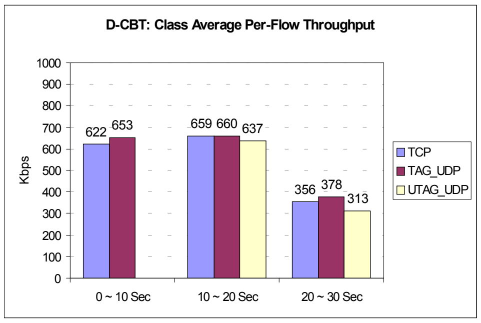
We measured the performance of RED, CBT and D-CBT in terms of fairness. In this section, we compare *the average per-flow throughput in each class*, which is an average aggregated class throughput divided by the number of flows in the class, to visualize how fairly the output bandwidth is assigned to each class considering the number of flows in the class. Also, later in this section we present the *Jain's fairness* [11] measurement results. Figure 11 (a) through (c) compares the periodic (i.e., 0-10, 10-20 and 20-30 seconds) average per-flow throughput for each class under the three queue mechanisms.



(a) RED



(b) CBT



(c) D-CBT

Figure 11: Average Per-Flow Throughput for TCP, Tagged UDP and Untagged UDP Classes under RED, CBT and D-CBT

As shown in Figure 11 (a), RED absolutely failed to assign bandwidth fairly to each class of flows from 10 seconds when the two high bandwidth untagged UDP flows (unresponsive CBR) joined transmitting at a total of 10Mbps, about 40% of the link bandwidth. During 0-10 seconds, when 25 TCP and 10 tagged (flow-controlled MM_APP) flows were competing for the bandwidth, it was somewhat unfair as a tagged flow got an average of 37% more bandwidth than a TCP flow, but RED was able to manage the bandwidth. However, when the untagged UDP blast came into the system, RED was totally unable to manage bandwidth. The 2 untagged UDP flows got most of the bandwidth they needed (average of 4.68Mbps out of 5Mbps), and the remaining flows used the leftover bandwidth. Especially, the 25 TCP flows got severely punished and transmitted at an average of 293Kbps per flow as they often went back to slow start and even timed out. Fairness got worse as 30 more TCP flows joined at 20 seconds and experienced starvation.

Figure 11 (b) shows that CBT can avoid the great unfairness of RED using fixed thresholds for the UDP classes. However, during the analysis, we found that CBT, which updates each class average and the RED average independently, suffers from *unsynchronized weighted-average updates*. That is, the ratio between independently updated UDP class averages and RED average does not correctly indicate the actual class bandwidth utilization ratio, since whichever flows update the average more frequently will have higher weighted-average than the others will, although they all use the same amount of bandwidth. This is because as the average is updated more frequently, not only is a newly enqueued packet added to the average with a predetermined weight, but also the existence of the other already enqueued packet are added to the average.

Figure 11 (c) shows the D-CBT results, which indicates that D-CBT fairly manages bandwidth during all periods accommodating the change of traffic mix by dynamically allocating the right amount of output queue space to each flow class. It also shows that by updating each class and the RED average at the same time in a synchronized manner, the ratio between the averages is a good indicator of the ratios between each class' bandwidth utilization. Note that although we strictly regulate the untagged class by assigning a fair threshold calculated from RED's minimum threshold, the untagged class did get most of its share. This is because the high bandwidth untagged (unresponsive) packets were allowed to enter the queue without a threshold test, when RED indicated no congestion.

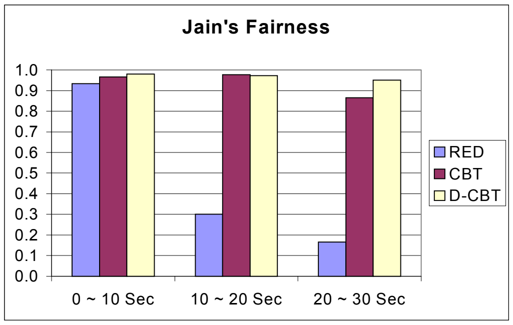


Figure 12: Jain's Fairness Comparison

Next, we present the same simulated systems' fairness on individual flows using *Jain's fairness index* [11], which measures the fairness among individual flows. Jain's equation takes the average throughputs of the flows of which the fairness is measured as input, and produces a normalized number between 0 and 1, where 0 indicates the greatest unfairness and 1 indicates the greatest fairness. Figure 12 shows the periodic (0-10, 10-20 and 20-30) Jain's fairness computation for each system.

Jain's fairness measurement shows that the simulated system that uses RED queue management fails to fairly assign bandwidth to each individual flow from 10 seconds when the unresponsive flows join in the system. The low Jain's index value for the RED system indicates that some flows are experiencing severe starvation during 10-20 seconds and even more severe starvation during 20-30 seconds when 30 extra TCP flows join.

The system that uses the CBT queue management mechanism was fair overall in distributing bandwidth to each flow. However, during 20-30 seconds, the system's fairness was degraded because the 10 tagged (multimedia) flows got about twice as much bandwidth as the other flows. One thing to note is that CBT's fairness was pre-engineered. In a circumstance where traffic mixes change a lot, CBT might show more degraded fairness.

On the other hand, the systems that use D-CBT were dynamically adjusting to changing flow mixes, and were very fair not only to the classes of flows but also to individual flows as Jain's index numbers indicate. However, this does not mean that D-CBT can force fairness within a class. Note that flows in a class used same flow control mechanism in the simulation (TCP Reno, MM_APP and Unresponsive UDP for each of the three classes). Also the distance from sources to the congested router were the same so that every connection in a class had very similar fragile (or robust) characteristics. D-CBT is designed to regulate fairness among the classes of flows using per-class accounting and is not aimed at forcing fairness within a class since per-flow accounting is expensive and may not scale well.

8. Conclusion

We have presented the design and evaluation of our proposed router queue mechanism, Dynamic Class-Based Threshold (D-CBT), by comparing its performance with that of RED and CBT. D-CBT is a new active queue management mechanism that addresses the problem of fairness by grouping flows into TCP, tagged (flow-controlled multimedia) UDP and untagged (other) UDP classes and regulating the average queue usage of the UDP classes to their fair shares.

As expected, RED, previously shown to be fair among TCP flows, showed an extreme unfairness with mixed traffic. CBT that uses a fixed threshold on UDP classes was able to avoid extreme unfairness. However, during the analysis, we found that CBT suffers from "unsynchronized weighted-average updates". D-CBT fixes CBT's problem by synchronizing all the average updates, and better manages bandwidth by dynamically determining the UDP thresholds to cooperate with RED by fairly assigning the output bandwidth to each class for all traffic mixes. That is, through class-based accounting, D-CBT fairly protects TCP from the effect of UDP flows and also fairly protects tagged UDP flows from untagged flows.

There exist many possible areas for future work and still remain many performance aspects to be evaluated. Future work could compare the performance of the D-CBT with that of FRED [7]. We expect that D-CBT could give better throughput performance for tagged UDP flows than FRED, since it frees flow-controlled multimedia flows from the strict per-flow punishment. Another area for future study is to examine the parameter sensitivity of D-CBT. We expect that D-CBT is as sensitive to parameter settings such as min-threshold, max-threshold and max-probability as RED, since its congestion detection and management is based on RED and only uses RED parameters. Another area for future work is to measure the effect of the threshold test of D-CBT on multimedia QoS with currently available responsive multimedia applications, since bursty multimedia packet drops when the class average reaches the class threshold may degrade the multimedia quality noticeably.

Recently, we implemented CBT and D-CBT into the Linux kernel, which currently works both for IPv4 and IPv6. Refer to [16] for the details. Related future work is to compare the Linux implementation of CBT and D-CBT with the NS implementations by reconstructing the simulated test runs. Additional future work is to measure and analyze the overheads of D-CBT using the Linux implantation and to optimize it.

9. Notes

The NS simulator code used in this research, CBT, D-CBT and the multimedia traffic generators, can all be downloaded from the perform web page, http://perform.wpi.edu/. The implementation of D-CBT in Linux will also be available shortly.

10. References

[1] Multimedia Communications Forum, Inc. “Multimedia Communications Quality of Service”, *MMCF/95-010, Approved Rev 1.0*, 1995, URL: http://www.luxcom.com/library/2000/mm_qos/qos.htm

[2] Claypool, M. and Tanner, J., “The Effects of Jitter on the Perceptual Quality of Video”, *ACM Multimedia Conference*, Volume 2, Orlando, FL, October 30 - November 5, 1999

[3] Floyd, S. and Jacobson, V., “Link-sharing and Resource Management Models for Packet Networks”, *IEEE/ACM Transactions on Networking*, Vol. 3 No. 4, August 1995

[4] Parris, M., Jeffay, K. and Smith, F. D., “Lightweight Active Router-Queue Management for Multimedia Networking”, *Multimedia Computing and Networking*, SPIE Proceedings Series, Vol. 3020, San Jose, CA, January 1999

[5] Floyd, S., “TCP and Explicit Congestion Notification”, *Computer Communication Review*, October 1994

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

[7] Lin, D. and Morris R., “Dynamics of Random Early Detection”, In *Proceedings of SIGCOMM ’97*, Cannes, France, September 1997

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

[9] VINT, “Virtual InterNetwork Testbed, A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB”, URL: http://netweb.usc.edu/vint

[10] Chung, J. and Claypool, M., “Better-Behaved, Better-Performing Multimedia Networking”, *SCS Euromedia Conference*, Antwerp, Belgium, May 8-10, 2000

[11] Jain, R., “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling”, John Wiley & Sons, Inc., New York, NY, 1991

[12] Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R. G., Krone, O., Sandvoss, J. and Vogt, C., “Media Scaling for Audiovisual Communication with the Heidelberg Transport System”, In *Proceedings of the Conference on Multimedia '93*, Anaheim, CA, August 1993

[13] Floyd, S. and Fall, K., “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, *Computer Communication Review*, V.26 N.3, July 1996

[14] Mitchell, J.L., Pennebaker, W.B., Fogg, C.E. and LeGall, D.J., “MPEG Video Compression Standard”, In *Digital Multimedia Standards Series*, Chapman & Hall, New York, NY, 1997

[15] Walpole, J., Koster R., Cen, S., Cowan, C., Maier, D., McNamee, D., Pu, C., Steere, D. and Yu, L., “A Player for Adaptive MPEG Video Streaming over the Internet”, In *Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-97*, SPIE, Washington DC, October 1997

[16] Leazard, N., Maldonado M., Mercado, E., Chung, J. and Claypool, M., “Class-Based Router Queue Management for Linux”, *Technical Report WPI-CS-TR-00-15*, Computer Science, Worcester Polytechnic Institute, April 2000