Dynamic-CBT and ChIPS – Router Support for Improved Multimedia Performance on the Internet

Jae Chung and Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
1-508-831-5357
{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), keep per-flow state and punish misbehaved, non-TCP flows. Unfortunately, these punishment mechanisms also result in a significant performance drop for multimedia flows that use flow control. We extend Class-Based Threshold (CBT) [12], 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. Also, as an effort to reduce multimedia jitter, we propose a lightweight packet scheduling called Cut-In Packet Scheduling (ChIPS) as an alternative to FIFO packet scheduling. The performance of our proposed mechanisms is measured, analyzed and compared with other mechanisms (RED and CBT) in terms of throughput, fairness and multimedia jitter through simulation using NS. The study shows that D-CBT improves fairness among different classes of flows and ChIPS improves multimedia jitter without degrading fairness.

Keywords

Multimedia, Router, Queue management, Congestion, Fairness, Jitter

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, specifically *delay* constraints and variance in delay, or *jitter* constraints [3,11]. 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.

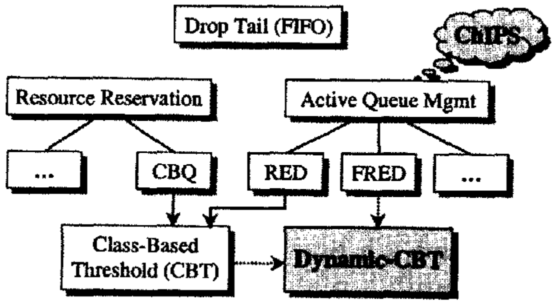


Figure 1. Router Queue Mechanisms (shaded are proposed)

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 such as DRR [13] and the Class-Based Queuing (CBQ) algorithm [7]. 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 [12].

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 [4]. 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 reveals the 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 [10,12]. In the worst case, it is possible for non-TCP flows, especially for 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.

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] [5]. Fair Random Early Drop (FRED) is an active queue management approach that incorporates this argument [10]. 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, FRED 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.

[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*.

Another major concern with FRED is that the per-active-flow accounting is expensive and might not scale well.

Core-Stateless Fair Queueing (CSFQ) [14] addresses the scalability problem by deploying hierarchical distribution of the per-flow accounting workload. In CSFQ, edge routers are distinguished from core routers, where edge routers calculate per-flow state information in terms of rate estimates and label outgoing packets, and core routers use the rate estimates to achieve fair allocation of the output bandwidth without maintaining per-flow states. The accuracy of CSFQ relies heavily on the assumption that all packets from a flow travel along the same path.

Jeffay *et al.*, [12] 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 effect 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. In addition, as a means to improve multimedia jitter, we propose a lightweight multimedia-favored packet scheduling mechanism, ***Cut-In Packet Scheduling*** (ChIPS), as an alternative to FIFO packet scheduling under D-CBT and possibly under other RED like active queue management mechanisms. ChIPS monitors average enqueue rates of tagged and the other flows, and is invoked when the tagged flows are using a relatively smaller fraction of bandwidth than the TCP flows. On transient congestion in which the queue length is greater than the average queue length, ChIPS awards well-behaved (flow-controlled) multimedia flows by allowing their packets to "cut" in the queue to the average queue length.

[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).

To evaluate the proposed mechanisms, we use an event driven network simulator called NS (version 2) that simulates a variety of IP networks [15]. NS implements most of the common IP network components including RED. We implement CBT in NS, extend it to D-CBT, add ChIPS into D-CBT, and compare the performance of D-CBT and D-CBT with ChIPS 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, and the effect of queue management on the performance of well-behaved (flow-controlled) multimedia flows.

2. PROPOSED MECHANISMS

This section presents the design and implementation of Dynamic-CBT (D-CBT) and Cut-In Packet Scheduling (ChIPS) in detail. Before describing D-CBT and ChIPS, we present the design of Class-Based Threshold (CBT) [12] 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. The conceptual view of the first CBT design is shown in Figure 2.

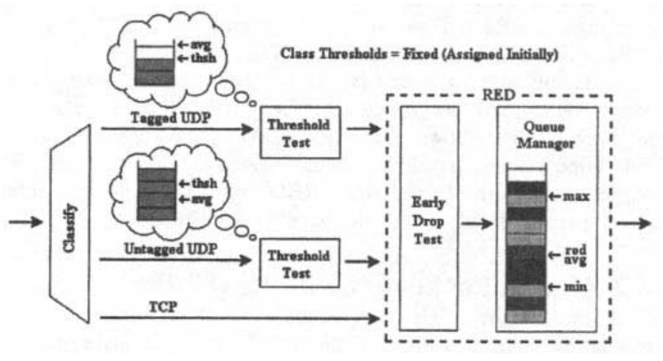


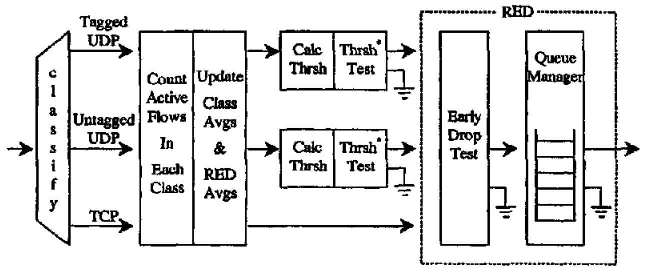
Figure 2. CBT (with RED for all) Conceptual View

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 only using 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 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.

2.1 Dynamic-CBT (D-CBT)

D-CBT enforces fairness among classes of flows, and gives UDP classes better queuing resource utilization. Figure 3 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.

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. 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, which may be different than those of 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.



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

Figure 3. Design of Dynamic-CBT (D-CBT)

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

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 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.

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 [10]. 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.

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 flow information and decreases the flow counter. The objective behind this probabilistic active flow counting approach is twofold: First, D-CBT does not necessarily require an exact count of active flows as do other queue mechanisms that are based on flow-based-accounting, although a more exact count is better for exercising fairness among flow classes. Second, 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 inserting and updating complexity of $O(n)$, and 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 operation drops to $O(1)$, while the complexity of the traverse delete will remain $O(n)$. Future work suggests more thorough measurement of added overhead.

When an incoming packet is updated or inserted according to its flow identification to 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 3. 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 share in average. 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. Second, assuming that the flows in a class (especially the tagged UDP flows) use flow control mechanisms of which the congestion responsiveness characteristics are almost the same, 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.

2.2 Cut-In Packet Scheduling (ChIPS)

ChIPS is a light-weight multimedia favored packet scheduling mechanism that can replace the FCFS enqueue style packet scheduling of a RED-managed queue for CBT, D-CBT and possibly other RED-like mechanisms, which is specifically targeted to improve multimedia jitter. ChIPS monitors the average enqueue rates of tagged and the other flows, and is activated when the tagged flows are using a relatively smaller fraction of bandwidth than the TCP flows. On transient congestion in which the queue length is greater than the average queue length, ChIPS awards tagged (flow-controlled multimedia) flows by allowing their packets to "cut" in the line of queue to the average queue length. Figure 4 shows the design of ChIPS.

By inserting tagged UDP packets at the average queue length on transient congestion, ChIPS improves flow-controlled multimedia jitter. However, this could harm the TCP flows and even make them time out by introducing a large extra delay when the multimedia traffic is taking a considerable portion of the output bandwidth. Under the normal RED queue mechanism that has no means to regulate the queue buffer usage among the classes of flows, it is essential for ChIPS to monitor the average enqueue ratio between the tagged and other flows and turn on its function only when the ratio is small. However, under CBT, in which the tagged threshold can be explicitly set to use a small fraction of the available queue buffer, this automatic turn on/off function is not really necessary. When used with D-CBT, the ratio that turns off ChIPS could be set relatively large (tested for up to 50% in our simulations with a RED minimum threshold of 5 and the maximum of 15) without degrading the fairness because of the "self-adjusting" ability of D-CBT. When a relatively large number of tagged flows compete for bandwidth with TCP flows, ChIPS could instantly lower the throughput of the TCP flows. However, this will also lower the average queue length of the queue, and therefore the fair threshold for the tagged class will be reduced and the tagged class throughput will be reduced as well. Thus, ChIPS may cause the average queue length to fluctuate a bit more but should not reduce fairness significantly. Section 5 has detailed results.

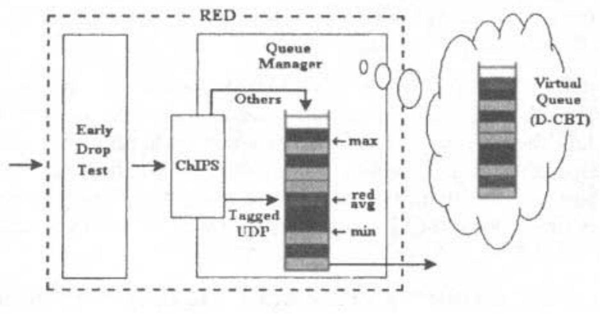


Figure 4. Design of ChIPS (Tagged Packet Insertion on Transient Congestion)

Another issue in implementing ChIPS is that the increment of the tagged packet dequeue rate caused by the insertion could degrade the fairness when the packet enqueue decision makes use of each class' buffer usage as in CBT and D-CBT. This faster tagged packet drain rate is not an issue for RED since its enqueue decision has nothing to do with the drain rate. However, in CBT and D-CBT, the faster drain rate lowers the average number of enqueued packets for the tagged class, which could result in the tagged class getting more bandwidth than its fair share. To prevent this effect, we used a virtual FIFO queue for counting the number of enqueued packets for the UDP classes, in which the class information of an enqueuing packet is always enqueued at the end, even though ChIPS cuts a tagged packet in the line of the real queue. In this way, the virtual queue can help more fairly count the class averages by telling if the tagged packets that have been transmitted already are still in the queue. Thus, the actual tagged packet drain rate does not affect the calculation of the average number of enqueued packets for the tagged class.

Looking at the complexity of the design, ChIPS has *O(1)* behavior, since the insertion complexity is *O(1)* and the virtual queue maintenance complexity is also *O(1)*. We believe that ChIPS, which noticeably improves tagged flow (flow-controlled multimedia) jitter, along with D-CBT would further encourage multimedia applications to use a flow control mechanism. An important issue that is not addressed in this paper is how to monitor and tag the flow-controlled multimedia flows. This issue of packet marking is beyond the scope of this paper, but can be extended from research into Diffserv scenarios [1]. We believe that this job should be done at the Internet Service Provider (ISP) or at the local network management level at the gateways to the public networks, and leave the routers free from this issue. The next section presents the methods we used to evaluate D-CBT and ChIPS.

3. PERFORMANCE METRICS

This section presents the fairness and jitter measurement metrics used to evaluate our proposed mechanism. To measure the fairness among the three different classes of flows and also to visualize the system's fairness on individual flows, we use the following two metrics. The first metric is an indicator of how fairly the output bandwidth is assigned to each class considering the number of flows in the class, called the *direct comparison of the average per-flow throughput in each class*. This is an average aggregated class throughput divided by the number of flows in the class. As the second fairness measurement metric, *Jain's fairness index* is used to visualize the fairness among individual flows [8]. Figure 5 shows the formula that calculates Jain's fairness, which gets the average throughputs of the flows (x_i) of which the fairness is measured as an input, and produces a normalized number between 0 and 1, where 0 indicates the greatest unfairness and 1 indicates the greatest fairness.

$$f(x_0, x_1, x_2, \cdots, x_n) = \frac{\left(\sum_{i=0}^{n} x_i\right)^2}{n \sum_{i=0}^{n} x_i^2}$$

Figure 5. Jain's Fairness Index Equation

Another network performance factor we measure is multimedia stream jitter (Figure 6). Jitter can is primarily measured in one of two ways: *variance in inter-frame arrival time* at the receiver, and *variance in end-to-end delay*. While the former is a receiver-oriented observation on the variance, the latter is a more network-oriented observation of the variance. Measuring jitter as variance in inter-frame arrival time (ex, r2 – r1) is useful when a traffic source's frame transmission interval is fixed. However, it may not be a good measure of jitter when the transmission interval varies as in the case of flow-controlled multimedia applications which may not transmit a frame in response to congestion. Measuring jitter in terms of end-to-end delay (ex, r2 – s2) is more direct indicator of a system's performance on multimedia streams, since it eliminates the inter-frame transmission periods of the source.

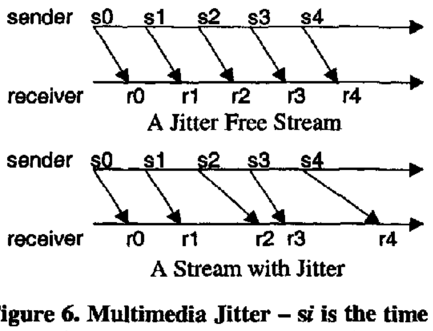


Figure 6. Multimedia Jitter – *si* is the time at which the sender transmits frame *i*. *ri* is the time at which the receiver receives frame *i*.

In real environments, it is hard to measure jitter in terms of variance in end-to-end delay because of asynchronized clocks at the source and destination. However, in our simulation environment where only one logical clock is used for the whole system, it is easy to measure the variance in the end-to-end delay. Moreover, this method can even visualize the effect of queuing delays of a single router on jitter well. Therefore, we measured jitter in terms of variance in end-to-end delay.

4. SIMULATION

We ran a simulation for each of RED, CBT, D-CBT and D-CBT with ChIPS. Every simulation had exactly the same settings except for the network routers, each of which was set to use one of the above four outbound queue management mechanisms. The network topology and the traffic source schedules are shown in Figure 7.

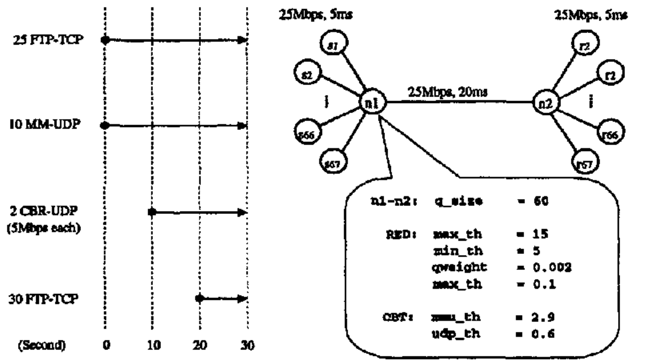


Figure 7. Simulation Scenario and Network Setup

In each simulation, we had 67 source nodes connected to one router and 67 destination nodes connected to the other router, which are interconnected by a link with 25Mbps bandwidth and 20ms of delay. Each link that connects a source (or destination) node and a router was set to have 25Mbps of bandwidth and 5ms of delay. For traffic sources, 55 FTP, 10 flow-controlled multimedia traffic generators called MM_APP [2] (tagged) and 2 CBR (untagged) 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 1Kbyte. The UDP agents were also set to have maximum packet size of 1Kbyte, so that all the packets in the network were the same size. 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 packet size of 1Kbyte. The CBR sources were set to generate 1Kbyte packets at a rate of 5Mbps.

We scheduled the traffic sources such that 25 TCP flows and 10 MM_APP flows were competing for the bandwidth during 0 to 10 seconds. At this period the fair bandwidth share for each connection was about 714Kbps (25Mbps / 35 flows). In the next period (10 to 20 seconds), the two high bandwidth CBR blasts joined trying to aggressively use the output bandwidth of which the average fair share was about 675Kbps (25Mbps / 37 flows). Later at 20 seconds, 30 more TCP flows came into the network lowering the average fair share during the last 10 seconds to about 373Kbps (25Mbps / 67 flows).

Network routers were assigned a 60-packet long physical outbound queue. The RED parameters, which are shown in Figure 7, 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 respectively to force each UDP flows to get about their fair bandwidth shares during 0 to 20 seconds. Assuming the average queue size is 10 packets, by reserving an average of a 2.9-packet space, the tagged class could get an average bandwidth of 7,250Kbps (25Mbps * 2.9 / 10) at congestion, which is about 10 times (10 tagged flows) the fair flow share during 0 to 10 seconds. Likewise, by reserving 0.6-packet space in the queue, the untagged class could get an average of 1,500Kbps, that is little bit more than 2 times (2 untagged flows) the fair flow share during 10 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. Finally, ChIPS was set to turn off its cut-in scheduling feature when the ratio between the number of tagged flows and the other flows are greater than 50%. However, under our simulation, ChIPS was always on since the ratio was always under 50%.

Thus, the simulations were designed to give an environment under which all three queue management mechanism manage output bandwidth fairness during the first 10 seconds, RED fails during the second 10 seconds, and CBT fails during the last 10 seconds. Then, we examine if D-CBT dynamically offers fair bandwidth allocation in every situation. Also, by comparing the results (fairness and jitter) of D-CBT with ChIPS with basic D-CBT, we examine the effect of ChIPS on fairness and multimedia jitter.

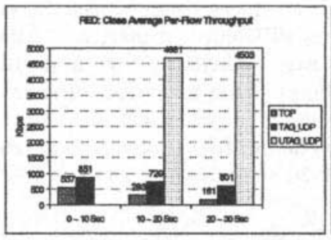
5. RESULTS AND ANALYSIS

We measured the performance of RED, CBT, D-CBT, and D-CBT with ChIPS in terms of fairness and multimedia jitter. We also compared TCP throughput under ChIPS and basic D-CBT as well as packet drop percentages.

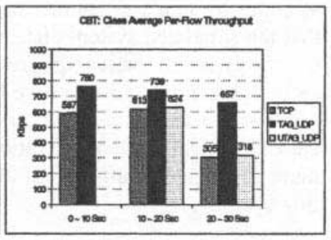
5.1 Class' Average Per-Flow Throughput

Figure 8 (a) through (d) compares the periodic (i.e., 0-10, 10-20 and 20-30 seconds) average per-flow throughput for each class under the four queue mechanisms.

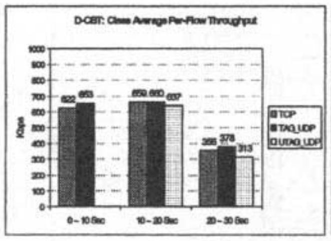
As shown in Figure 8 (a), RED absolutely failed to assign bandwidth fairly to each class of flows from 10 seconds when the two high bandwidth untagged UDP flows (unresponsive CBR) join transmitting at a total of 10Mbps, about 40% of the link bandwidth. During 0-10 seconds, when 25 TCP and 10 tagged (flow-controlled 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.



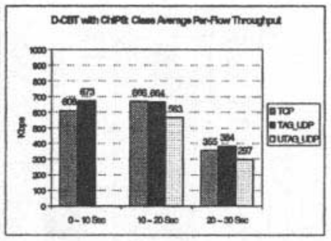
(a) RED



(b) CBT



(c) D-CBT



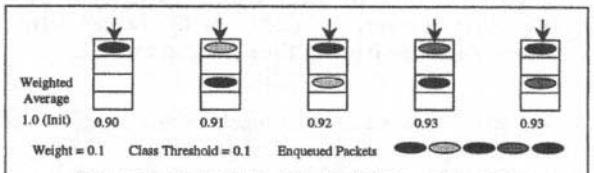
(d) D-CBT with ChIPS

Figure 8. Average Per-Flow Throughput for TCP, Tagged UDP and Untagged UDP Classes under RED, CBT, D-CBT and D-CBT with ChIPS

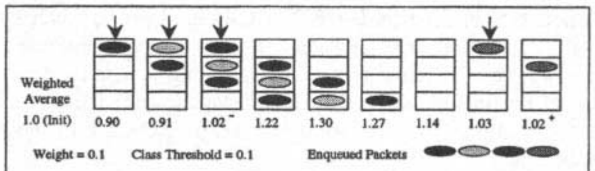
Figure 8 (b) shows that CBT can avoid the great unfairness of RED using fixed thresholds for the UDP classes. However, CBT was not assigning the output bandwidth to each class as expected. When designing the simulation, we set the UDP thresholds such that during 0-10 seconds each tagged UDP flow should get about 725Kbps on average. During 10-20 seconds, we expected that each tagged flow's average bandwidth would remain the same and each untagged UDP flow would get an average of 750Kbps. Also, we expected that during 20-30 seconds, the tagged and untagged flows would get a large portion of the bandwidth the same as during 10-20 seconds and the TCP flows would get much less than the fair share during this period. However, the simulation result shows that the tagged UDP class got more bandwidth than the expected values especially during the last period, while the untagged UDP class got much less bandwidth than expected.

We found that this is mainly due to how and when CBT updates each UDP class threshold and RED queue average. CBT updates each UDP class average only for incoming packets that belong to the class, and the RED unit updates its queue average for all incoming TCP packets and for UDP packets that passed an appropriate threshold test. Therefore, the class averages and the RED queue average are almost independently updated at different speeds that are closely related the number of incoming packets that belong to the class. In addition, the RED average has a higher chance of being updated faster than the UDP class averages. In this situation, which we call *unsynchronized weighted-average updates*, whoever (i.e. a class) updates its weighted-average more often will get less bandwidth by having a larger weighted-average than the average of others for the same amount of class output bandwidth, and the output bandwidth is controlled using the averages at the UDP threshold test units.

Figure 9 shows this effect by comparing two situations where a UDP class that has an initial class weighted-average of 1, a weight of 0.1 and a class threshold of 1.02, is experiencing two different incoming packet rates. Figure 9 (a) is the case when the incoming packet rate is 0.5 packets per packet transmission delay, and Figure 9 (b) is the case when the class is receiving packets at the rate of 1.0 packets per packet transmission delay. In this example, it is assumed that the traffic sources are unresponsive CBR applications. One thing to note in the figure is that the class average shown at the left bottom of each queue in each state is its value before making the enqueue admission decision for an incoming packet at that state. As you can see in the figure, as the number of incoming packets for a class increases, packets are enqueued in a bursty manner, and more importantly, its class average gets larger. As the average is updated more frequently, not only is a newly enqueued packet added to the average (with the weight of 0.1), but also the existence of the other already enqueued packet are added to the average. For example, the existence of the first packet is added to the average 2 times more for the second situation than for the first situation. Note that Figure 9 (a) enqueues more packets but has a lower class average.



(a) Incoming Packet Rate = 0.5 pkts / pkt-transmission-delay



(b) Incoming Packet Rate = 1.0 pkts / pkt-transmission-delay
(Dropped packets are not shown)

Figure 9. A CBT Class's *Weighted Average* under Two Different Incoming Packet Rates

The weighted-average calculation method works fine when the purpose of measuring an average queue size is to detect impending congestion as in RED. However, when the method is used to assign bandwidth to different classes of flows by comparing each class' weighted-average number of enqueued packets, we have determined that all the weighted-averages should be updated at the same time and at an equal frequency to give a correct output bandwidth utilization ratio among the classes. In the case of CBT, by measuring each UDP class average and the RED average independently, the classes' bandwidth utilization could not measured correctly by comparing the class averages. By comparing the fairness measurement in Figure 8 (b) and CBT's outbound queue averages in Figure 10, especially for 20-30 seconds, one can easily see that CBT's attempt of using unsynchronously updated weighted-averages to regulate class bandwidth was misleading. Figure 10 indicates that during 20-30 seconds, 10 tagged flows used an average of about 2.5 packet-spaces in the queue that is 0.25 packet-spaces per each flow, and the 2 untagged flow used an average of about 0.6 packet-spaces that is 0.3 packet-spaces per each flow. However, as shown in Figure 8 (b), each tagged flow used about 657 Kbps of bandwidth and each untagged flow used about 318 Kbps, about one half of the per-flow bandwidth of a tagged flow.

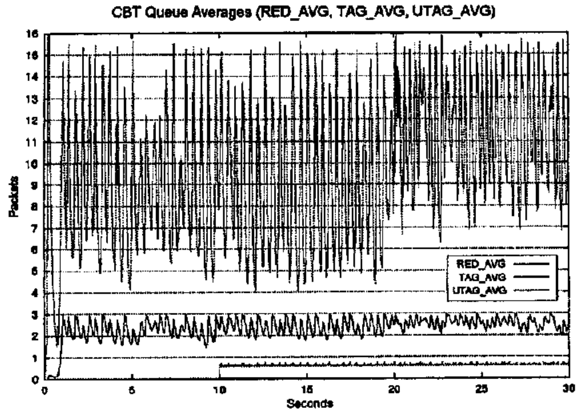


Figure 10. CBT Queue Averages – The top line is the RED average, the middle is the tagged UDP average and the lower is the untagged average.

From the above observation, we conclude that the current CBT design can only prevent a great unfairness caused by unresponsive or misbehaving flows, and it needs some adjustment on weighted-average calculation. Indeed, we tried the average calculation method that is used in D-CBT in CBT and got a much better result, that is the ratio between the three averages indicates the ratio between the actual classes' bandwidth utilization. However, we did not include the result in this paper, since the method is used in only D-CBT and we are presenting D-CBT in the next paragraph.

Figure 8 (c) shows the D-CBT results, which indicates that D-CBT fairly managed bandwidth during all periods by dynamically allocating the right amount of output queue space to each flow class. It also shows that by updating each class and 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. One thing to note in the figure is 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 8 (d) shows the result of D-CBT with ChIPS. The result confirms that ChIPS, when used with D-CBT, does not affect fairness between each class of flows, due to the virtual queue and D-CBT's self adjusting capability described in Section 2.2. In the simulation, the ratio between tagged flows and all flows was about 28% during 0-20 seconds, and was about 15% during the last 10 seconds.

5.2 Jain's Fairness Measurement

Figure 11 visualizes the simulated systems' fairness on individual flows using Jain's Fairness Index, where the periodic (0-10, 10-20 and 20-30) average throughput of each individual flow was given as input to Jain's equation. 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.

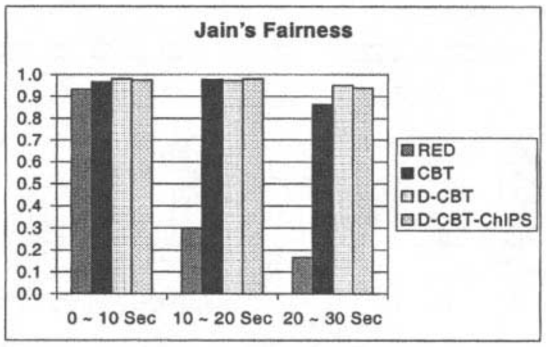


Figure 11. Jain's Fairness Comparison

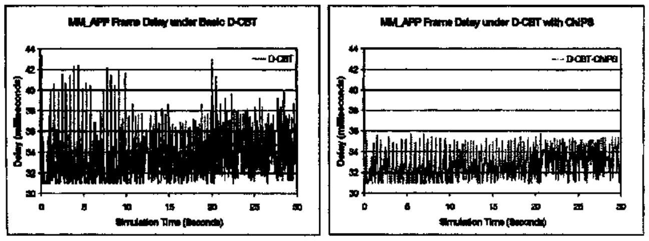
The system that uses the CBT queue management mechanism was fair overall in distributing bandwidth to each flow. However, during 20-30 seconds, the system's fairness was degraded because the 10 tagged (multimedia) flows got about twice as much bandwidth as the other flows. One thing to note is that CBT's 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 or D-CBT with ChIPS 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. Jain's fairness measurement results on D-CBT and D-CBT with ChIPS also reconfirmed that ChIPS did not degrade the system's fairness.

5.3 Analysis of ChIPS

Now that we have shown that D-CBT outperforms RED and CBT in managing bandwidth, and that the use of ChIPS does not degrade the performance of D-CBT, this section presents the performance of ChIPS on multimedia jitter and TCP throughput. Figure 12 shows tagged UDP (or multimedia) jitter by comparing a MM_APP application's end-to-end frame delay under D-CBT and D-CBT with ChIPS.

The result indicates that ChIPS does improve tagged stream jitter by inserting tagged packets into the line of the queue to which the RED average points on transient congestion. Under ChIPS, the maximum tagged-UDP jitter was about 5*ms* (36*ms* – 31*ms*) while it was about 12*ms* (43*ms* – 31*ms*) under normal D-CBT. Noting that the maximum threshold of RED was set to 15 packets, which gives about 3*ms* (15pkts * 8Kbits / 25Mbps) of queuing delay, ChIPS was able to regulate the maximum tagged stream jitter around the queuing delay of the RED's maximum threshold.



(a) Basic D-CBT (b) D-CBT w/ ChIPS

Figure 12. ChIPS Effect on Multimedia Jitter

We believe that ChIPS effect on improved multimedia jitter could be very significant because of the following two reasons. First, what we show in Figure 12 is the jitter gain due to a single router. When multiple routers are involved, the jitter gain due to ChIPS could be larger. Second, in the simulation, we used multimedia frames that are the same size as that of a network packet, meaning that no frame fragmentation occurs in the IP layer. Assuming that a multimedia application uses frames that are larger than a network packet and are chopped into multiple packets in the network, the jitter improvement due to ChIPS could be even more significant, since the multimedia packets have a better chance to be transmitted close to each other at routers. Thus, we believe that the potential for ChIPS to improve multimedia jitter is larger than shown in our experiments.

Table 1. TCP Packet Accounting (0 ~ 30 Seconds)

	TCP Packets Delivered	TCP Packet Drop Rate	TCP Throughput
D-CBT	66,648 pkts	4.46 %	17,773 Kbps
D-CBT w/ ChIPS	66,386 pkts	4.44 %	17,703 Kbps

Table 2. Tagged (MM) Packet Accounting (0 ~ 30 Seconds)

	Tagged Packets Delivered	Tagged Packet Drop Rate
D-CBT	21,126 pkts	11.85 %
D-CBT w/ ChIPS	21,519 pkts	12.95 %

Lastly, we present TCP packet accounting and tagged packet accounting for the simulation in Table 1 and Table 2. Table 1 shows the TCP packet drop rate and throughput under ChIPS is very compatible with those of basic D-CBT. The TCP throughput under ChIPS was about 99.6% of the throughput under basic D-CBT. This indicates that ChIPS, when used along with D-CBT, may not significantly affect the TCP throughput. Comparing the TCP throughput loss with the multimedia jitter gain, ChIPS compensates 14.3% ((42*ms* - 36*ms*) / 42*ms* * 100) of multimedia jitter gain for 0.4% of TCP throughput loss for the simulation.

Table 2 shows the multimedia packet drop rate of the system that used ChIPS is very compatible with that of the system that used basic D-CBT. This result shows that ChIPS has a high potential to improve end-user multimedia performance (perceptual quality) on the Internet by improving jitter without increasing the multimedia packet drop rate, which is another important factor in multimedia perceptual quality and for congestion control and system utilization.

6. CONCLUSION

In this paper, we have presented the design and evaluation of our proposed router queue mechanisms, Dynamic Class-Based Threshold (D-CBT) and Cut-In Packet Scheduling (ChIPS), by comparing their 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. ChIPS is a multimedia-favored lightweight packet scheduling mechanism that can substitute the FCFS enqueue style packet scheduling part of a RED-managed queue for D-CBT and possibly for other RED-like queue mechanisms.

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". 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 class updates the average more frequently will have higher weighted-average than the others will, although they all use the same amount of bandwidth.

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. We have also shown that ChIPS, when used with D-CBT, can improve multimedia jitter without degrading fairness.

There exist many possible areas for future work and still remain many performance aspects to be evaluated. Recently, we implemented CBT and D-CBT into the Linux kernel [9], which currently works both for IPv4 and IPv6. Our current ongoing work is in measuring and analyzing the overheads of D-CBT using the Linux implementation and in optimizing it. 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.

Another possible future project would be to extend this study to evaluate the limitation of ChIPS on the fairness and the link utilization offered by D-CBT. As noted in Section 2.2, ChIPS introduces an additional delay to other traffic which may affect TCP throughput. Therefore, in order for the use of ChIPS to be more practical, future work suggests an extended study to determine the maximum average ChIPS enqueue ratio between tagged and the other classes of flows without degrading fairness or link utilization. An additional project would be to evaluate D-CBT and ChIPS under the environment where fragile and robust

TCP connections as well as multimedia connections with different end-to-end delays coexist in the system. Another study that we could not do due to the lack of time but suggest as a future work is to compare the performance of the D-CBT with that of FRED. 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.

7. ACKNOWLEDGMENT

The authors are grateful for the detailed suggestions by several anonymous reviewers.

8. REFERENCES

[1] Bernet, Y. et. Al. "A Framework for Differentiated Services", February 1999, Internet Draft, draft-ietf-diffserv-framework-02.txt

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

[3] 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

[4] Floyd, S., "TCP and Explicit Congestion Notification", *Computer Communication Review*, October 1994

[5] Floyd, S. and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, February 1998

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

[7] 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

[8] 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

[9] 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

[10] Lin, D. and Morris R., "Dynamics of Random Early Detection", In *Proceedings of SIGCOMM '97*, Cannes, France, September 1997

[11] 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

[12] 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

[13] Shreedhar, M. and Varghese, G., "Efficient Fair Queueing using Deficit Round Robin", In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995

[14] Stoica, I., Shenker, S. and Zhang, H., "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks", In *Proceedings of SIGCOMM '98*, Vancouver, Canada, September 1998

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