# Responsive vs. Unresponsive Traffic: Active Queue Management for a Better-Than-Best-Effort Service

*Mark Parris  Kevin Jeffay  Don Smith*
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175 USA
*{parris,jeffay,smithfd}@cs.unc.edu*

**Abstract**: In the best-effort Internet, there is a fundamental tension between responsive and unresponsive flows. During periods of congestion, responsive flows reduce the load they generate while unresponsive flows may or may not do so. As a result, responsive flows can suffer from a starvation effect while unresponsive flows benefit from their greedy nature. Many of the proposed approaches to this problem focus on encouraging end-systems to use responsive protocols by deploying mechanisms in network switches that penalize unresponsive traffic. We argue that while there are applications that are unnecessarily unresponsive, there are also some classes of applications that have legitimate motivations for using unresponsive protocols. Based on this argument we approach the problem with the goal of isolating these three types of flows (responsive, unnecessarily unresponsive, and necessarily unresponsive) from each other.

We survey the problem of managing responsive and unresponsive traffic and propose an active queue management mechanism that insures responsive flows are not penalized for their cooperative nature while also insuring that necessarily unresponsive flows have acceptable performance. Our mechanism, called class-based thresholds (CBT), allocates buffer capacity within a switch's FIFO queue to each type of flow in proportion to the desired bandwidth allocation between traffic types. We demonstrate empirically (1) the starvation effect of unresponsive traffic on responsive traffic, (2) the effect of unresponsive traffic on proposed active queue management mechanisms, and (3) the ability of CBT to efficiently provide isolation between traffic classes and provide a better-than-best-effort service to necessarily unresponsive traffic.

## Responsive and Unresponsive Traffic

Responsive and unresponsive flows[1] are distinguished by their response to network congestion. Congestion occurs when the aggregate load exceeds the capacity on some bottleneck link. When congestion occurs, a queue builds up in the router servicing the bottleneck link. The initial effect is simply poor performance in the form of increased end-to-end latency. If the congestion persists, the queue overflows and packets are lost. The widely accepted appropriate response to this situation is to reduce the load in an attempt to match the available capacity of this bottleneck link. Protocols that follow this approach and use some form of *congestion-control* are referred to as *responsive*. The most common example of a responsive protocol is the Transport Control Protocol (TCP) [1]. Those protocols that fail to detect or choose to ignore congestion by simply maintaining (or even increasing!) their load are referred

[1] Here we use the term flow simply as a convenient way to designate the packets traveling between a source and destination. In the IP context, a flow is the collection of packets having a common addressing 5-tuple of *<source, source port, destination, destination port, and transport protocol type.>*

to as *unresponsive*. The most common example of an unresponsive protocol is the Unreliable Datagram Protocol (UDP) [1]. The most widely advocated responsive mechanism is to decrease the generated load geometrically when congestion is detected and increase load linearly in order to probe for available capacity when the congestion subsides [2]. Because this approach was first shown to be effective in TCP, those protocols that closely follow this approach are referred to as TCP-friendly. This congestion response mechanism allows the system to reach equilibrium with the generated load approximately equal to the available capacity of the bottleneck link. Moreover, since all TCP-friendly sources respond to loss in the same manner, all of the sources should be impacted equally by congestion and thus network resources are shared in loosely fair manner. In principle, if all sources reduce their load during a period of congestion, the overload is averted.

Unresponsive protocols, intentionally or unintentionally, abuse the cooperative nature of responsive traffic. Sources using unresponsive protocols do not decrease their load in response to congestion indicators such as packet loss. As a result, congestion persists unless responsive sources continue to decrease their load below their fair share. The responsive sources stabilize their aggregate load at the difference between the true capacity of the bottleneck link and the load generated by the unresponsive sources. As a result, the unresponsive flows are able to use the lesser of their maximal bandwidth requirement or the capacity of the link. Thus in the extreme, unresponsive flows can starve responsive flows of bandwidth.

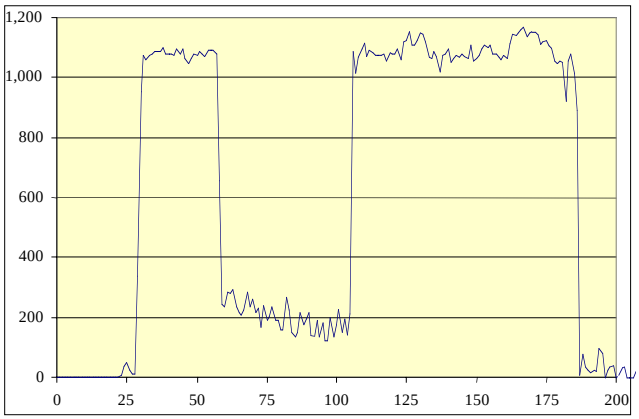



**Figure 1**: Responsive Traffic in the Presence of Unresponsive Traffic (KB/s vs. time).

Figure 1 illustrates the starvation of responsive traffic. This figure shows the results of an experiment performed on a small internetwork wherein two simulated campus networks are connected via a 10 Mbps link. The plot shows the aggregate throughput of responsive traffic (TCP) over the 10 Mbps link. A large number (thousands) of TCP flows are introduced at approximately time 25. The flows quickly stabilize and are able to share the full capacity of the link. High-bandwidth unresponsive (UDP) traffic is introduced at time 60, and deactivated at time 110. Although the same application load is present on the end-systems, the responsive traffic's throughput decreases from 1,100 KB/s to 200 KB/s while the unresponsive traffic is present. While the unresponsive sources benefit from their uncooperative behavior, the responsive flows suffer poor performance. This phenomenon may encourage application designers to use unresponsive protocols in an effort to receive better performance during periods of congestion. Recognizing the danger presented by this scenario, the Internet community is widely advocating mechanisms to encourage the use of end-to-end congestion control [3],[4]. They encourage the deployment of network mechanisms that will provide a disincentive to use unresponsive protocols by dynamically identifying and aggressively penalizing unresponsive flows during periods of congestion. However, many of the approaches proposed seem to assume unresponsive sources could be modified to behave in a TCP-friendly manner. As we argue next, this is not likely to be the case for all flows.

**The Nature of Responsive and Unresponsive Flows**

TCP and UDP are the most common examples of responsive and unresponsive protocols. In fact, 95% of the bytes traversing the Internet use the TCP protocol while UDP traffic makes up most of the remaining 5% [5]. As such, we use TCP and UDP as the representative examples of each type of traffic. In order to appreciate why applications choose responsive (TCP) and unresponsive (UDP) protocols it is helpful first to examine each of these protocols more closely, focusing on their primary features as well as their response to congestion.

The primary feature of TCP is the abstraction of a reliable, ordered byte-stream. As long as connectivity is maintained, all data entering the stream at the source will arrive at the receiving application in the order it was sent. TCP thus works well for applications that require reliable in-ordered delivery such as file-transfers or web page loads. To provide reliability the protocol requires the receiver to acknowledge receipt of each packet of data. If an acknowledgement is not received in a reasonable period of time after the packet is transmitted, the source infers that the original packet was lost and retransmits the packet. In TCP's original design a source simply retransmitted lost packets while maintaining its rate of transmission of new data. Given that the primary cause of loss in the Internet is the overflow of a router's queue because of overload, responding to losses by only retransmitting data leads to a pathological network condition known as *congestion collapse*. If all of sources respond to loss by retransmitting while maintaining their load, congestion persists at the bottleneck link and loss continues. Moreover, the overload at the bottleneck link leads to inefficient use of the upstream links as the lost data traverses those links multiple times (because of the retransmissions).

In response to this problem, researchers added congestion-avoidance mechanisms to TCP. TCP geometrically decreases the amount of data allowed to be in transit at any one time (its window size) when losses are detected and linearly increases the window size to probe for available capacity when losses subside. Using this control-loop, TCP attempts to find equilibrium between network capacity and offered load. Note that TCP infers congestion by the implicit detection of losses resulting from the absence of acknowledgements. There is no explicit mechanism to detect congestion. Instead, TCP leverages the existing feedback mechanism already present for reliability to infer the presence of congestion. It is important to further note that responsiveness requires some form of feedback mechanism in order for the source to detect congestion.

In contrast to TCP, UDP is an unreliable transport protocol. UDP provides a simple best-effort datagram service. Application data units are packaged into datagrams and passed to the network protocol but there is neither assurance nor confirmation that the datagram reaches the intended receiver. UDP is best suited for applications that can tolerate loss, and where each unit of data is relatively independent. UDP is often used to transmit data that is periodically superseded, such as audio or video frames. In those applications loss of one datagram is not a concern because the information in a subsequent datagram supersedes the lost datagram. One of UDP's primary strengths is its simplicity and the resulting low overhead. Because the protocol is connectionless and unreliable, there is no overhead associated with maintaining connection state or with storing packets for possible retransmission until the destination confirms their receipt. Moreover, as soon as a packet arrives at the destination, it is available to the application layer. There are no delays such as the ordering delays incurred while waiting for the retransmission of a lost or mis-ordered packet. As a result, those packets that do arrive successfully at the destination do so with low end-to-end latency. Here we use end-to-end latency to refer to the time between the time the sending application passes the packet to the transport layer and the time when the packet becomes available to the receiving application. Note that there is no feedback mechanism at the transport layer for UDP. Without such a feedback mechanism it is difficult to extend UDP to detect and, consequently, respond to congestion.

Because TCP-friendly protocols are vulnerable to aggressive, unresponsive flows, there are incentives to deploy mechanisms to encourage the use of end-to-end congestion control mechanisms. Most of these mechanisms take a TCP-centric approach, discouraging and penalizing flows that do not use end-to-end congestion control mechanisms in an effort to encourage them to adopt a TCP-friendly protocol. However, it is important to consider why applications are built using responsive or unresponsive protocols. In most cases the issue of responsiveness has very little to do with the application designer's choice of protocol. Application designers are concerned with higher level properties of the protocol. For example, recall that TCP's fundamental feature is reliability. Application designers usually choose to use TCP not because of its responsiveness, but because they need to ensure the integrity of the data being transferred. However, this integrity comes at a cost. Packets may have to be retransmitted multiple times before reaching the receiver. This means that other packets may be delayed while waiting for a prior packet in the byte stream to be retransmitted. Although all data will eventually reach the receiver, very little can be said about its timeliness. For many applications this is acceptable. Web and file-transfers that take 30 seconds are just as correct as those that take 30 milliseconds. Although a smaller response time is always preferable, it does not impact the integrity of the data or the correctness of the application.

Other applications, such as videoconferencing and IP telephony, do have minimum requirements for timeliness and bandwidth. If the end-to-end latency is too high, interactivity in these applications is lost. Moreover, one must typically maintain some minimal frame- or sample-rate to ensure minimal fidelity of the communication. These application level concerns translate into a minimum transmission rate requirement. Additionally, these so-called continuous media applications are able to tolerate some packet loss without the need for retransmissions. Because frames are encoded as independent units (or independent groups), losses are also independent at the application level. A packet loss may result in the loss of a frame or group of frames, but it does not compromise the correctness of other groups of frames. For example, because video frames must be played out periodically to give the illusion of motion, the loss of a few frames may be below the threshold of perception of the user. Rather than freezing the play-out while waiting for the retransmission of the $n$-th frame, the application can simply skip to the $n$+1-th frame and continue on. Because continuous media can tolerate some loss and is more concerned with minimizing end-to-end latency, application designers choose unreliable protocols with low overhead, like UDP. Moreover, they specifically avoid reliable protocols like TCP because reliability through retransmission introduces a great deal of variability in the end-to-end latency while offering a feature that is not required. Those applications that do have concerns about limiting loss may elect to employ application-level techniques for ameliorating the effects of loss such as the use of adaptive media encoding [6, 18]. However, none of this is to say that multimedia applications are inherently unresponsive. Rather, because frame-rate and end-to-end latency are major concerns, application designers usually choose to respond to congestion by adjusting the amount of data they try to send, not simply the rate at which they send it. To do this, application-level techniques are used that take advantage of knowledge of the media encoding and semantics. For example, an application may use temporal (e.g. frame-rate) or spatial (e.g. image resolution) scaling techniques to adjust the quality of the media stream [6]. Nonetheless, the point remains: application designers seldom choose protocols based on their responsiveness. They choose protocols that come closest to offering the services and performance they require. Beyond the issue of responsiveness, what is actually needed is a "better" service than the current best-effort delivery service of the Internet.

In addition to application level approaches there has been a good deal of work focused on new unresponsive and "semi-responsive" protocols which incorporate feedback for the purpose of making streaming media applications more responsive, both at the application and transport level [7],[8]. These approaches show promise, however, until they are fully developed and widely deployed, the tension between responsive and unresponsive traffic remains a problem. In our work we focus on network-centric approaches where the problem is addressed without changing the end-systems but, rather, through

changes to network infrastructure. Specifically, we focus on approaches in an area of router design called active queue management.

**Active Queue Management**

Active queue management refers to the practice of manipulating queues in routers to bias the performance of network connections. For our purposes here we focus on the process of being proactive in deciding which packets to discard from the queue and when to discard them. For example, one approach, Random Early Detection, focuses on giving better feedback to responsive flows to indicate when congestion is imminent by monitoring the average queue occupancy and probabilistically dropping packets before the queue overflows [9]. Another approach, Flow Random Early Detection, seeks to insure equal sharing of link bandwidth between flows by managing queue occupancy on a per flow basis [10]. Other work has focused on identifying and penalizing unresponsive flows by monitoring the queue behavior [4]. Here we summarize some of the active queue management approaches and introduce our approach, class-based thresholds. We will then compare the performance of each of these algorithms in the presence of unresponsive traffic in an attempt to understand the impact that active queue management mechanisms can have on application performance.

*Drop-Tail FIFO*

In today's Internet, packet loss from router queues is the primary indicator of congestion. Routers have packet queues associated with each outbound link. These queues are intended to buffer bursty packet arrivals before forwarding on the outbound interface. In most Internet routers, these queues are simple FIFO queues with *drop-tail* discard semantics. Arriving packets are enqueued at the tail of the queue. When the queue fills, the arriving packet is discarded. The "drop-tail" behavior is effective in providing congestion notification to responsive flows as demonstrated by the success of the Internet. However, with drop-tail, the decision to drop a packet is essentially a passive one. Moreover, drop-tail has several flaws that prompted research into a more active approach to router queue management [3]. These flaws are most apparent during periods of persistent congestion and include the problems of *lock-out* and *full queues*. Briefly, lock-out refers to a phenomenon where a few flows are able to monopolize the queue space. Because of TCP synchronization effects, packets from some flows always arrive to a full-queue and are subsequently dropped, effectively locking those flows out of the outbound link and preventing them from making progress. (For more details, see [3].) Full queues also occur during persistent congestion. When the queue is consistently full, it is simply a source of latency and cannot serve its intended function of buffering bursty arrivals. Moreover, because drop-tail routers only drop packets when the queue is full, sources are only able to detect and respond to congestion after it has grown persistent. Notification of imminent congestion would allow sources to activate their avoidance mechanisms before congestion becomes severe.

Active queue management seeks to remedy these, and other problems with drop-tail FIFO queuing and provide TCP better feedback to respond to.

*RED*

The most well known and widely deployed active queue management mechanism is Random Early Detection (RED). The main goal of RED is to provide better feedback to responsive flows. This goal has several parts. First, RED seeks to do a better job detecting the onset of congestion instead of waiting until congestion is persistent and the queue is overflowing. Second, RED seeks to distribute feedback more evenly across all flows. Instead of the disparity in drops that can lead to the lock-out phenomenon observed with drop-tail, RED seeks to insure that all flows have an equal percentage of their packets dropped. Finally, RED seeks to maintain shorter average queue occupancy. The intent is to avoid full queues and have queue space available to accommodate bursty arrivals, even during periods of modest congestion.

RED accomplishes these goals by actively monitoring and managing the queue. Since a queue builds up when the offered load exceeds the link's capacity and the queue drains when the load is less than the link's capacity, a measure of the queue's length over time is a good indicator of the state of congestion. Using the *instantaneous* queue occupancy is problematic as it may be the result of recent bursty arrivals and not persistent congestion. Thus, instead of using instantaneous queue occupancy as a congestion indicator, RED uses the recent *average* queue occupancy. The RED algorithm maintains a running weighted average of queue occupancy. When the average is below a minimum threshold value, packets are simply enqueued and forwarded. When the average exceeds the minimum threshold, arriving packets are randomly dropped. The probability the arriving packet will be dropped is a function of the average queue occupancy and the number of packets that have arrived since the last packet was dropped. The probabilistic component of this mechanism allows RED to distribute the drops more evenly across all flows. Better distributed drops mean better feedback to all flows, allowing them all to back-off equally. Finally, dropping packets before the queue fills helps to maintain a shorter average queue.

RED has proven effective in addressing its stated goals. As a result, RFC2309 recommends that some form of active queue management, specifically RED, be widely deployed in routers [3]. However, neither RED nor drop-tail FIFO addresses the tension between responsive and unresponsive flows. (Neither was intended to address this problem; each dealt with other significant problems: accommodating bursts and providing better feedback to responsive flows.) As a result, the same RFC also recognizes the tension between responsive and unresponsive flows and encourages continued research in how to deal with flows that are unresponsive or not TCP-friendly. Most research in this area has focused on approaches that are *TCP-centric*. That is, they focus on identifying and penalizing, often severely, unresponsive traffic in an effort to encourage the use of responsive protocols. We discuss two such approaches below. The first is another active queue management scheme, Flow Random Early Detection [10], focused on providing fair sharing of bandwidth using per flow statistics. The other approach outlines techniques for identifying different classes of unresponsive traffic [4].

*FRED*

FRED seeks to insure that all flows receive a fair share of the link's capacity. To do this the algorithm logically manages the queue on a per-flow basis. FRED maintains statistics for every flow that has a packet enqueued in the router. During periods of congestion (as determined by monitoring the average queue size) the algorithm drops packets for those flows that occupy more than twice their fair share of the queue. A fair-share of the queue is $1/n^{th}$ of the average queue occupancy where $n$ is the number of flows with a packet enqueued. The limit is *twice* a fair share to accommodate burstiness. However, once a flow has violated this loose fair share limit, it has a "strike" recorded against it and is strictly constrained to occupy no more than exactly its fair share. This constraint is lifted only when the flow has no packets enqueued. Since high-bandwidth unresponsive flows will maintain their load, they will usually have a packet enqueued and this constraint will be maintained continuously. Conversely, if a responsive flow happens to generate a burst of packets that triggers the strike mechanism, it should respond to the resulting drop by reducing its load, leading to a period when it has no packet enqueued at the bottleneck router, allowing the strike to be erased. This strike mechanism should tightly constrain unresponsive flows while allowing responsive flows to get a larger share of the link capacity. (For additional details see [10].)

FRED provides a mechanism that offers fairness by managing queue occupancy on a per-flow basis. This approach is effective in constraining individual, unresponsive flows and addressing the tension between responsive and unresponsive flows. However, this solution requires significant overhead as state must be maintained for each active flow. Moreover, it is not clear that equal bandwidth shares is the best kind of fairness. Continuous media applications often have some minimum bandwidth requirement while file-transfers are primarily concerned with data integrity and response time is a secondary concern. In some situations dividing link capacity could result in slightly faster response times for data transfers while

making an interactive video-conference useless. Consider an example where a video-conference with a minimum bandwidth requirement of 1.5 Mb/s shares a 10Mb link with 19 file-transfers using TCP. If the video-conference receives its minimum requirement, the file-transfers can share the remaining 8.5 Mb/s equally. However, if FRED constrains all flows to equal shares, the file-transfers receive 11% more of the link capacity while the video-conference will be constrained to one-third of its minimum requirement, making the interaction useless. The file-transfers' integrity is the same in either case, though the response time is 11% better using FRED. However, the video-conference has value in the former case and no value in the latter.

*Detecting Unresponsive Flows*

Other work has focused on extending existing active queue management mechanisms such as RED for detecting different classes of unresponsive flows. Floyd and Fall propose tests for detecting these different types of flows by monitoring the RED drop-histories of different flows [4]. With this technique they can identify which flows are TCP-friendly, high-bandwidth, or unresponsive. They then propose that those flows that are not TCP-friendly be constrained in an effort to encourage application designers to use TCP-friendly protocols. This and similar approaches focus on better feedback for responsive flows and recommend actively discouraging unresponsive flows to promote the use of end-to-end congestion control.

**CBT**

In our approach, class-based thresholds (CBT), we recognize the importance of protecting responsive traffic from the effects of unresponsive flows, but we also recognize there are applications that have legitimate reasons for using unresponsive protocols [11]. As such, we seek a solution that protects responsive traffic while also insuring that necessarily unresponsive traffic continues to make progress. Instead of seeking to severely penalize unresponsive traffic, we seek to isolate responsive traffic from the effects of the unresponsive traffic. Moreover, we try to isolate necessarily unresponsive traffic from other unresponsive traffic, and insure that the necessarily unresponsive traffic can claim a managed share of the link's capacity. At one extreme, one could provide bandwidth allocation using packet scheduling mechanisms to insure each class receives exactly its desired service rate [12]. However, for reasons of efficiency and simplicity, we seek to explore how effectively we can approximate this type of resource allocation by using thresholds on the occupancy of a single queue instead of packet scheduling. In the implementation of our work presented here, we identify three classes of traffic: responsive (TCP), tagged (necessarily unresponsive traffic such as continuous media), and "other" (everything else). We then use thresholds on the average queue occupancy of each class of traffic to effectively allocate the capacity of the bottleneck link between these traffic types. The limits on average queue occupancy serve two functions. First, they determine the proportions of the queue's capacity available to each class. Since this queue feeds the outbound link, these proportions also hold on the outbound link. Second, the occupancy thresholds in aggregate determine the maximum average queue occupancy. The maximum average queue occupancy, in turn, establishes a limit on the maximum average queue-induced latency. This is an important consideration as we try to insure low latency for continuous media.

It's important to note that this algorithm does *not* maintain separate queues for each class. It simply maintains separate state for each class and bases the drop decision for an arriving packet on the statistics for that class. It is also worth noting that there are far fewer traffic classes (3!) than active flows (typically thousands). Moreover, while tagged and other traffic have a single threshold to simply limit the queue occupancy, the algorithm uses a RED algorithm with a minimum and maximum threshold for the responsive traffic. As a result, responsive traffic continues to receive early notification of imminent congestion but only if the responsive traffic is the source of the overload. If traffic in the tagged or other class is the source of the overload, then packets from those classes are dropped and responsive traffic behaves as if the link is uncongested. The ratio between the thresholds for the classes determines the ratio

of bandwidth allocated for each class during periods of congestion. However, the thresholds equate to allocations, not upper limits, for bandwidth for each class. The algorithm implicitly allows for borrowing link capacity in a *min-max fair* manner [13]. If some class is using less than its allocation, other classes borrow that class's leftover capacity in proportion to their thresholds.

CBT provides an active queue management facility to allocate bandwidth to multiple classes of traffic. Using this approach we can isolate responsive traffic from the effects or both necessarily and unnecessarily unresponsive traffic. Moreover, we can isolate the two classes of unresponsive traffic from each other. CBT also allows one to control latency directly by (deterministically) controlling the maximum queue length. Combined these features can result a better-than-best-effort service for the necessarily unresponsive traffic.

**Packet Scheduling - CBQ**

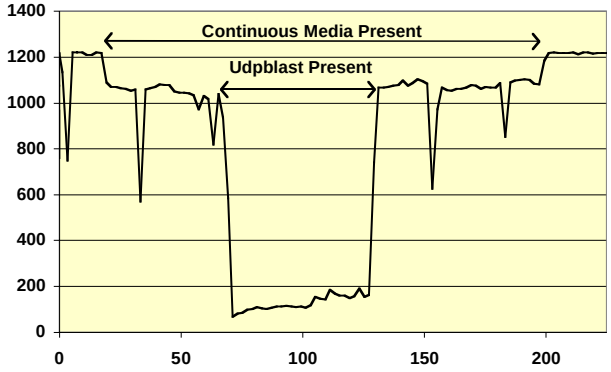
Finally, for comparison purposes, we also consider an approach to bandwidth allocation and isolation based on packet scheduling. Rather than deciding which packets to discard when, a packet scheduler divides different classes of traffic into separate queues and services each queue at a specified rate. In this manner a scheduler is capable of providing perfect isolation between traffic classes (since packets are enqueued by class) and perfect bandwidth allocation (since queues can be serviced at precise rates). However, this quality of service comes at the price of significant complexity and state in the router. The router must maintain multiple queues and perform a (potentially non-trivial) computation prior to dequeuing every packet in order to maintain the desired service rates. We seek to determine how close we can come to the performance offered by packet scheduling using only queue management.

We use the packet-scheduling algorithm, Class-based Queues (CBQ) [12], as a baseline to which we compare the performance of the different active queue management algorithms. We configure CBQ to classify packets into the same traffic classes used in CBT and to service queues according to a pre-determined share of the link capacity we have chosen to allocate to each traffic class.

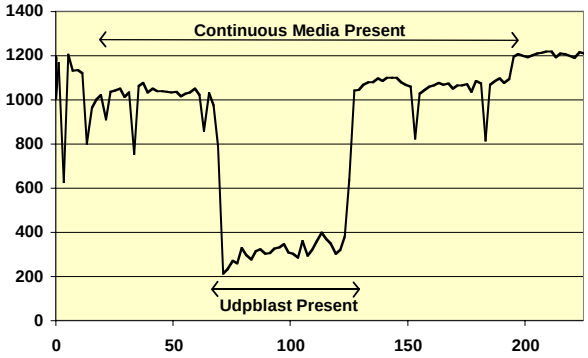
**Empirical Comparison**

We have conducted a series of experiments using the network infrastructure described in [15], to demonstrate the effectiveness of each active queue management scheme above in addressing the tension between responsive and unresponsive flows. In each of these experiments we have a bottleneck router (a PC running FreeBSD) using either one of the ALTQ [14] implementations of drop-tail FIFO, RED, or CBQ or our own implementation of FRED or CBT. (To the best of our knowledge there are no implementations of the mechanisms described in [4].) We tuned each algorithm to offer optimal performance based on the algorithm's intended goals. For FIFO, RED, and FRED we focused on TCP performance while for CBT and CBQ we considered the performance of all the traffic types. To generate TCP traffic we simulate a large collection (thousands) of users browsing the Web using an application-model of Web traffic [16]. To avoid synchronization effects, we introduce artificial delays at each receiver using *dummynet* [17]. Multimedia traffic is generated with an application based model of the Intel ProShare videoconferencing system [18]. We also generate aggressive unresponsive traffic with an application generating UDP traffic at a rate sufficient to saturate the bottleneck link. In each experiment, we measured TCP throughput by monitoring the bottleneck link using *tcpdump*. We measure multimedia loss and latency by instrumenting our traffic generators.

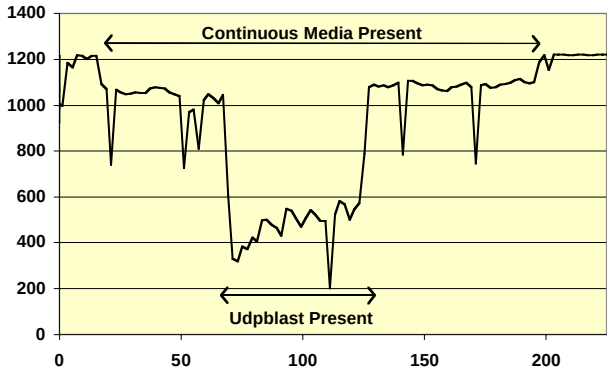
For each algorithm, we begin by introducing enough responsive traffic (Web traffic using TCP) to saturate the bottleneck link. We then introduce 6 "tagged" multimedia streams with an aggregate load of ~160 KB/s. Finally, we introduce aggressive unresponsive traffic (referred to as a "udpblast"). Finally, we stop the udpblast, multimedia, and responsive traffic, in that order. The period each traffic type is present is indicated on the relevant plots.



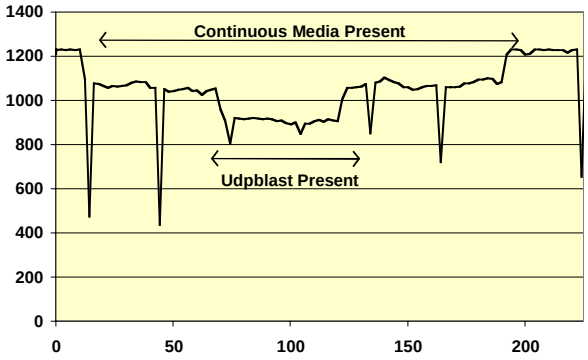

*a*) DROP-TAIL



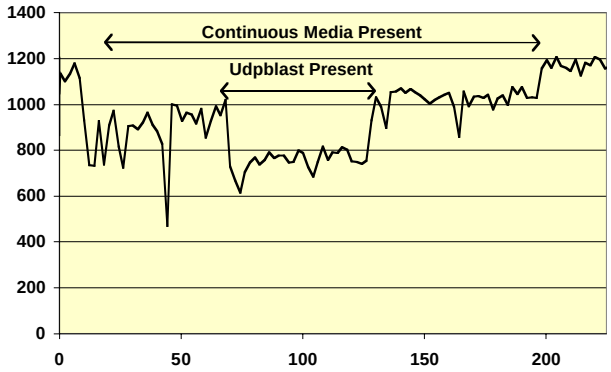

*b*) RED.




*c)* FRED.




*d*) CBQ.




*e)* CBT.

**Figure 2**: Aggregate TCP throughput under DROP-TAIL, RED, FRED, CBQ, and CBT (TCP throughput in kilobytes/second versus elapsed time in seconds.)[2]

[2] To the Editors: We can merge these plots into a single unit if we the plots will be reproduced in color.

To evaluate the effect of the various active queue management mechanisms, we first consider the throughput of responsive traffic. Figure 2 compares the throughput of TCP under drop-tail FIFO, RED, FRED, CBT, and CBQ. Note that in the case of drop-tail FIFO and RED, TCP throughput collapses when the aggressive flows are introduced. This is because the TCP flows respond to the resulting congestion and packet loss by decreasing their load while the aggressive flows continue to transmit at the same rate. Since the TCP load is reduced, the aggressive flows are able to consume the unused bandwidth (the bandwidth no longer consumed by TCP) and dominate the link. In contrast, FRED better polices the unresponsive flows and results in better TCP throughput. FRED's per-flow fairness constrains the aggressive flow and allows TCP to continue to make progress. The CBQ experiment shows the best possible performance that can be expected. By scheduling packets at guaranteed rates, CBQ provides a guaranteed service for those classes that operate within their allocations. (In this experiment, CBQ was configured to allocate 74% of the link's capacity to TCP, 14% to multimedia, and 12% to "other.") Before the aggressive flows are introduced TCP is able to consume more than that share, but its throughput reduces to its allocated level when the udpblast is introduced. Finally, we show the performance under CBT where CBT's thresholds are set to provide multimedia with its desired 160 KB/s, other traffic with 150 KB/s, and responsive traffic with the remaining capacity. TCP's throughput is much better than RED, FRED, or drop-tail FIFO and nearly comparable to that of CBQ. However, the more interesting fact is that CBT is able to achieve this TCP throughput with out compromising the performance of the necessarily unresponsive traffic, as we will see when we consider latency and loss for those flows.

Except for CBT, none of the active queue management schemes give special consideration to multimedia or other unresponsive flows that have minimum acceptable performance levels. In the case of RED and FRED, there was no effort to distinguish between flows so multimedia was simply as likely to receive drops as any other traffic type. Table 1 shows the multimedia drop-rate during the udpblast period. Clearly, RED forces a large number of drops for continuous media, with a 31% drop-rate. Performance under drop-tail FIFO is similar. (Generally speaking, for the class of interactive continuous media applications we are considering, loss rates much above 5% are likely to render the application useless.) FRED's policy of restricting unresponsive flows results in a similar drop-rate for continuous media, 27%. These drop-rates are unacceptable if the multimedia interaction is to have any value. In contrast, a well-configured CBT offers a drop-rate of 3%. This is comparable to the performance of the packet scheduling discipline. CBQ guarantees no drops as long as the class's offered load remains within its allocated bandwidth and since that is the case here we see a drop-rate of 0% for continuous media.

Finally, we look at the average network latency of multimedia flows for the various queuing and scheduling schemes. If interactivity is to be maintained the queue-induced latency at each router must be minimized. Drop-tail FIFO gives the worst performance as it allows the queue to remain full during periods of congestion and the average latency is directly related to the queue size. In the case of RED and FRED, the maximum threshold parameter limits the average queue occupancy and is the primary constraint on queue size. A maximum threshold of 30 packets in the case of RED results in latency on the order of 30 ms. FRED requires a larger queue threshold in order to maintain a reasonable drop rate. As a result, the average latency is almost 80ms. In contrast CBT's threshold settings are tuned to have a lower limit on the maximum average queue size, resulting in an average latency of 20ms. Finally, since CBQ schedules packet transmissions, any class that is within its bandwidth allocation and non-bursty should be serviced almost immediately. In this case, the overall latency is 7ms for multimedia.

**Table 1:** Average Packet Drop Rate and Latency for Multimedia Packets

| Queue Management Scheme | Drop Rate for Continuous Media | Latency |
|---|---|---|
| DROP-TAIL | 38% | ~ 54 ms |
| RED | 31% | ~ 35 ms |
| FRED | 27% | ~ 79 ms |
| CBT | 3% | ~ 20 ms |
| CBQ | 0% | ~ 8 ms |

**Summary**

In the Internet today there exists a fundamental tension between protecting TCP connections from congestive collapse and supporting the needs of multimedia applications that require continuous transmission of data. The Internet as a whole requires TCP, or TCP-friendly, reactions to congestion in order to operate effectively. Unresponsive traffic, however, including well-behaved multimedia, has the potential to take advantage of the cooperative nature of responsive flows and dominate a link's capacity while forcing responsive flows to decrease the load they generate. The current focus of the Internet research community is on the use of active queue management mechanisms to resolve this tension in favor of TCP performance and at the expense of even necessarily-unresponsive traffic. We have argued that useful and important applications exist which are inherently unresponsive and have sought to demonstrate that this tension can be resolved in a manner that is more amicable to both responsive and necessarily unresponsive traffic classes.

We have surveyed two of the basic classes of active queue management, RED and FRED, and have proposed a third method called CBT. RED focuses on providing probabilistic early notification of impending congestion to responsive flows and as such is ineffective against isolating TCP from unresponsive connections. FRED extends RED by promoting fair-sharing of link capacity through minimal and maximal allocations of queue capacity that are a function of the number of flows present in the router. CBT on the other hand uses static allocation of capacity to a small (and configurable) number of traffic classes to both isolate traffic classes from one another, and to ensure well-behaved non-responsive traffic realizes *better* service than under simple drop-tail FIFO (or RED or FRED).

Through a series of experiments we have shown:

- Congestion collapse is possible (and easy!) under simple drop-tail FIFO or RED. This demonstrates that the tension between response and unresponsive traffic classes is real and needs to be addressed.
- TCP receives a larger share of available bandwidth when facing unresponsive flows under CBT than with RED, FRED, or drop-tail FIFO. Performance is comparable to that achieved with a guaranteed service packet scheduler such as CBQ but with significantly less state and simpler mechanisms.
- Under CBT, the number of drops experienced by low-bandwidth multimedia flows is substantially lower than when RED or FRED is used and within the range that can be accommodated by application-level error control schemes.
- Under CBT, end-to-end multimedia latency can be tuned to a value lower than that observed with RED or FRED.

While CBT is not a panacea and issues such as marking schemes for packets remain, we believe it demonstrates that the goals of providing a better-than-best-effort forwarding service for well-behaved unresponsive traffic and protecting responsive traffic from unresponsive traffic are not inherently contradictory.

**References**

[1] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*. Addison-Wesley, Reading, Mass. 1994.

[2] V. Jacobson, Congestion Avoidance and Control, *ACM Computer Communications Review*, 18(4):314-329, Proceedings of ACM SIGCOMM '88, Stanford, CA, August 1988.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, & L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", *Internet draft, work in progress*, 1998.

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

[5] K. Claffy, G. Miller, K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone", http://www.caida.org/outreach/papers/Inet98/

[6] Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R., Krone, O., Sandvoss, J., Vogt, C., 1993. "Media Scaling for Audiovisual Communication with the Heidelberg Transport System", *Proc. ACM Multimedia '93*, Anaheim, CA, August 1993, pp. 99-104.

[7] S. Cen, C. Pu, J. Walpole, "Flow and Congestion Control for Internet Streaming Applications", *Proc. SPIE/ACM Multimedia Computing and Networking '98*, San Jose, CA, January 1998, pages 250-264.

[8] D. Sisalem, H. Schulzrinne, The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme, *Eighth International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Cambridge, UK, July 1998, pp. 215-226.

[9] S. Floyd, & V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Trans. on Networking*, V.1 N.4, August 1993, p. 397-413.

[10] D. Lin & R. Morris, Dynamics of Random Early Detection, *Proc. SIGCOMM '97*.

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

[12] S. Floyd & V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks", *IEEE/ACM Transactions on Networking*, V.1, N.4, August 1995, pp. 365-386.

[13] S. Keshav, *An Engineering Approach to Computer Networks: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, professional computing series, 1997

[14] K. Cho, "A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers", *USENIX '98, Annual Technical Conference*, New Orleans, LA, June 1998.

[15] M. Christiansen, K. Jeffay, D. Ott, F. D. Smith, "Tuning Red for Web Traffic", *ACM SIGCOMM 2000*, Stockholm, Sweden, August, 2000.

[16] B. A. Mah, An Empirical Model of HTTP Network Traffic, in *Proceedings of the Conference on Computer Communications (IEEE Infocomm)*, (Kobe, Japan), pp. 592-600, Apr. 1997.

[17] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", *ACM Computer Communication Review*, January 1997

[18] P. Nee, K. Jeffay, G. Danneels, "The Performance of Two-Dimensional Media Scaling for Internet Videoconferencing," Proceedings of the Seventh International Workshop on Network and Operating System Support for Digital Audio and Video, St. Louis, MO, May 1997, pages 237-248.