Computer Networks 40 (2002) 19–43

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

# Improving the performance of interactive TCP applications using service differentiation [☆]

Waël Noureddine, Fouad Tobagi [*]

*Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA*

**Abstract**

Interactive TCP applications, such as Telnet and the Web, are particularly sensitive to network congestion. Indeed, congestion-induced queueing and packet loss can be a significant cause of large delays and variability, thereby decreasing user-perceived quality. We consider addressing these effects using service differentiation, by giving priority to interactive applications' traffic in the network. We study different packet marking schemes and handling mechanisms (packet dropping and scheduling) in the network. For marking packets, two approaches are considered. First, we look into application-based marking, and show how the protection of Telnet traffic against loss can eliminate large echo delays caused by retransmit timeouts, and how, by limiting packet loss for Web page downloads, their delays can be significantly reduced, resulting in enhanced interactivity. Second, we consider differentiation based on TCP state, where we present a marking algorithm that prioritizes packets at the source, based on each connection's window size. In addition, we describe the shaping mechanisms required for conformance to agreements with the network. We show how this marking results in good response times for short transfers, which are characteristic of interactive applications, without significantly affecting longer ones.
© 2002 Published by Elsevier Science B.V. All rights reserved.

*Keywords:* TCP applications; Service differentiation; TCP-state based differentiation; User-perceived performance

## 1. Introduction

We have all had the frustrating experience of dealing with large and variable delays when using interactive Internet applications, such as Telnet and the Web. These applications clearly have more stringent delay requirements than "traditional" data applications like FTP and email. For example, human-computer interaction studies have shown that the response time of highly interactive tasks (such as teletyping in Telnet), should be below 150 ms for best user-perceived performance [33]. Beyond that, delays in response time (e.g., Telnet echo delays) become noticeable and, eventually, they would severely hinder the usability of the application, especially if delay variability increases as well. Comparably stringent constraints apply to other highly interactive data applications, such as remote graphical desktop access and real-time gaming. Similarly, Web page downloads should complete in a few seconds (e.g., less than 5 s [7]), and should have low variability to be

[☆] A shorter version of this paper appears in [30].
[*] Corresponding author.
*E-mail addresses:* noureddine@stanford.edu (W. Noureddine), tobagi@stanford.edu (F. Tobagi).

satisfactory to users. The low delay and high predictability requirements have also been found to depend on the perceived importance of the page content and the task at hand. For example, they are stricter for business applications, such as e-commerce and online trading, than for normal Web browsing (for more information on user-perceived performance of interactive applications, the reader is referred to [7,9,33] and the references therein). The growing importance of these and similar Internet applications' role in our daily life behooves us to improve their delay performance.

Delays in response time are introduced in the network as well as in the servers. Clearly, heavily loaded servers may introduce large delays in response time for interactive (e.g., Web) transfers. A content provider interested in decreasing these delays can do so by increasing server capacity (e.g., using higher performance hardware), by prioritizing requests based on the application or the importance of the request for interactivity [15], or by using content replication and caching. In contrast, network delays, which form a significant part of total delay for Web transfers [4,5,25,27], are usually outside the control of the provider or any other single organization, and thus not as easily reduced. In this paper, our focus is on network delays, and we assume that server performance has been properly addressed and server delays are therefore negligible.

For a concrete example of the impact of network delays on interactive applications, consider Telnet. In the common usage of Telnet, users type characters at a terminal, at speeds up to five characters per second [33]. These are sent over a TCP connection to a server, which echoes them back. Network delay for Telnet is the time between typing a character and the reception of the corresponding character echo. It includes transmission, propagation and queueing in network buffers. Telnet is sensitive to *per-packet* delays, and therefore these components can perceptibly affect the end-user experience. Furthermore, if the packet containing the character or the echo is dropped in the network, additional delays are introduced as TCP's reliability mechanisms are invoked to recover the lost data.

Similarly, network delay for Web browsing is the time between the generation of a page request and the reception of the corresponding Web page components (HTML code and in-lined images). [1] Again, this delay includes transmission, propagation and queueing delays for individual packets. However, the delays due to TCPs mechanisms for connection establishment, reliability and congestion avoidance and control are typically the most significant. This is particularly the case for HTTP/1.0, where a TCP connection is opened for each component of a page, adding a non-negligible connection establishment overhead to the total transaction delay. As discussed in [27], the use of one, "persistent", TCP connection to transfer all Web requests and responses between a client and a server eliminates this overhead. This usage has been adopted in HTTP/1.1.

We are interested here in the delays due to network congestion-induced queueing and packet loss. TCP was designed with the goal of realizing the maximum *throughput* over a path with unknown bandwidth and round trip delay. During a long transfer, TCP actively probes the network for available resources by continuously increasing its window and therefore the amount of data it injects in the network, filling up network buffers until packet loss occurs. Packet loss is followed by a period of idle time, and a possibly severe reduction of the sending window. Such loss, and the time needed for recovery typically do not significantly affect the long term average throughput of a large transfer. However, the impact of large delays in queues and packet drops for interactive transfers that share the same network buffers is significant. Indeed, the delays thereby introduced are excessive for *delay sensitive* applications, and result in degradation of user-perceived performance. While bottlenecks may not exist in the reputedly over-provisioned backbone of the Internet, they tend to naturally occur along the paths of connections, for example at the boundary between different service providers' networks, or between wired and wireless

[1] To simplify the presentation, we ignore DNS lookup delays. However, we note that the mechanisms we study should also be used to decrease the network component of these delays.

networks (which typically have limited bandwidth resources). Given the burstiness of TCP traffic and the uncontrolled usage of the network, congestion and packet loss are bound to occur at these bottlenecks. Therefore, when examining response time for interactive TCP applications, there is a strong motivation to address the effects of network delays due to congestion.

In this paper, we achieve the goal of reducing congestion-induced delays for interactive applications using service differentiation mechanisms, such as those defined in the IETF *DiffServ* architecture (see [8]), and in the *Assured Forwarding* service in particular. We consider two approaches to the use of these mechanisms. In the first, preferential treatment is given to interactive applications in the network, thereby reducing the packet loss rate they incur. Thus, highly interactive applications, such as Telnet, would be given priority over interactive applications, such as Web transfers, which in turn are given higher priority over non-interactive applications. We show that, by properly classifying traffic based on the applications' characteristics and requirements, user-perceived quality can be significantly improved, albeit at the expense of lower priority traffic. The second approach automatically prioritizes short (interactive) transfers by basing the priority of packets on the TCP connection window. A source marking algorithm is described, which allows fine-grain control on the performance of individual connections. This approach is shown to improve the user-perceived performance of interactive transfers, without significantly affecting others.

The rest of this paper is organized as follows. Section 2 describes the simulation setup used in the study. Section 3 motivates the work, by illustrating the effects of congestion on Web page downloads and Telnet echoes. In Section 4, we present the service differentiation framework assumed for the study. We describe the different network functions that are expected in edge and core routers, and in source hosts. In Section 5, we show how prioritizing interactive applications traffic in the network can improve the performance of such applications. Limitations in this approach lead us to look for a more flexible solution. In Section 6, we propose a set of generic TCP state-based service differentiation mechanisms that can be used to improve the performance of all TCP applications. We conclude in Section 7.

## 2. Simulation setup

This study relies on computer simulations, using *ns* [1]. Therefore, we pay particular attention to the design of an accurate and realistic simulation setup, which we describe in this section, justifying the choices made along the way. Unless otherwise noted, the parameters specified below were used for all the experiments in the paper.

### 2.1. Network scenario

To illustrate the issues at hand, it is sufficient to consider one network bottleneck, shared by all connections. We therefore use a symmetric, multihop tree topology, shown in Fig. 1, where sources and destinations are communicating across the bottleneck. We use typical link speeds, starting from the Users side: 1.5 Mbps (e.g., T1), 10 Mbps (e.g., 10 Mbps Ethernet) and 45 Mbps (e.g., T3). The bottleneck link speed is varied in the scenarios. Given the relatively high speed links chosen, and in order to generate a realistic traffic aggregate, several hundred traffic sources of the different types are needed. Furthermore, to capture the effects of the aggregation of many flows, which may modify the characteristics of individual flows, traffic from different sources is aggregated at several points before reaching the bottleneck. The topology thus contains a total of 800 hosts, organized in 400 source–destination pairs, as follows: ten users are connected to each 1st level router, eight 1st level routers are connected to each 2nd level router, and five 2nd level routers are connected to each bottleneck router. We have also experimented with different topologies, fewer users and correspondingly lower link speeds, with similar results.

The simulated network only needs to capture the main aggregation points and potential bottlenecks of a larger, more complex network. Therefore, each link in the topology effectively represents several actual links, as well as the

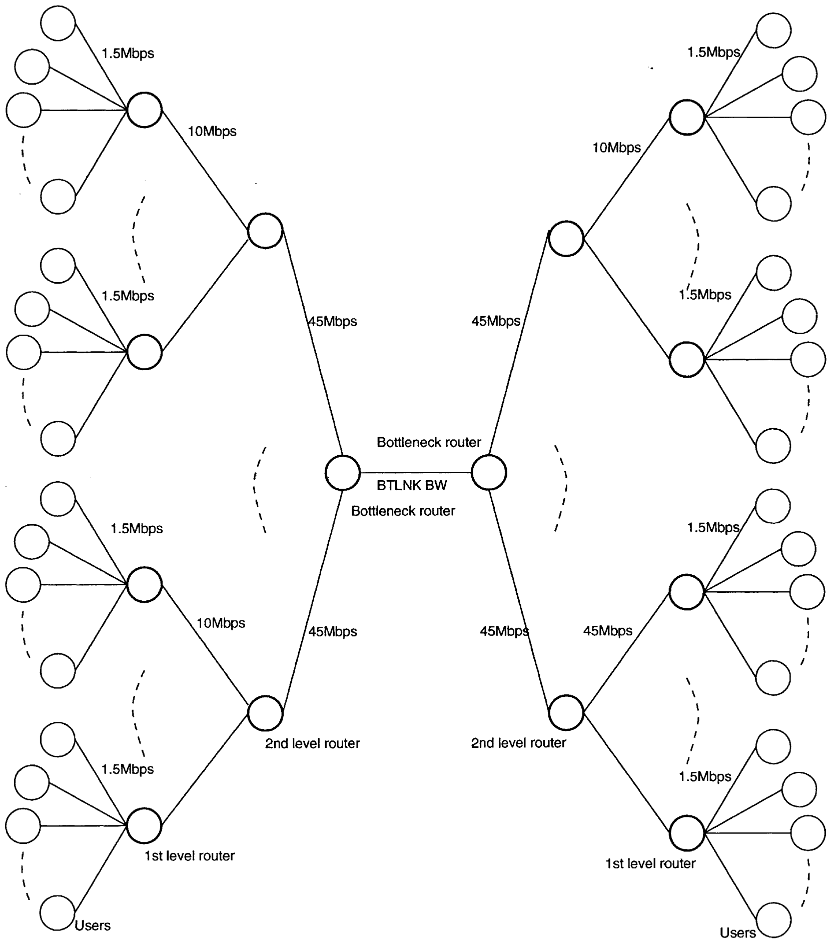


Fig. 1. Network topology.

intermediate nodes. Hence, the propagation delay of each link in the simulation accounts for the transmission and propagation delays on the links it represents, and the switching delay in the intermediate nodes. The delays for the different links in the topology are selected to lead to a mix of round trip times between different source–destination pairs (20, 40, 80, 120 and 200 ms), thereby covering a wide range of RTTs, from metropolitan to inter-continental.

In order to generate network congestion at levels similar to those seen in the Internet, and since the number of flows in the simulation is limited, we use buffers that are smaller than what is common in commercial equipment. On the 1.5, 10, 45 Mbps and bottleneck links they are 64, 64, 250, and 500 packets, respectively.

### 2.2. *Traffic models*

The simulation results presented in this paper use TCP NewReno. However, the same experiments were repeated for the Reno and SACK versions, and identical results were obtained. In order to remove the limitation of small receiver advertisement on the sending window size, and therefore emphasize the more interesting role of the congestion window, the receive buffer size was set to 64 KB (the maximum unscaled value). However, given the congestion levels seen in the simulations, a buffer size of 32 KB (a more common value, used by Linux receivers) would have produced identical results. Smaller values (e.g., 8 or 16 KB) would have limited the sending rate of some sources in scenarios with large bottleneck link speed and affected, to some extent, the numerical values obtained.

We model the following representative TCP applications traffic: *interactive* Web, Telnet and FTP, generated in proportions that attempt to *roughly* approximate their real life counterparts, across the range of bottleneck links used.

### 2.2.1. HTTP

We use two different HTTP models, one for HTTP/1.0 and the other for HTTP/1.1. The HTTP/1.0 client sends a request which, when completed, is followed by the server sending the HTML index page. When the index is received, up to four connections are opened in parallel to transfer the objects (e.g., images) embedded in the page, as in popular commercial browsers. After each object is received, the corresponding connection is closed, and a new one opened if more objects remain to be transferred. In contrast, the HTTP/1.1 server uses only one "persistent" connection to send all the objects assuming a pipelined request, i.e., all object requests are considered to be received together and therefore all objects are sent without inter-object delay. The connection is closed when the transfer is complete. The performance measure we use, *download time*, is the delay from the time a request is sent, until the whole page is received.

The composition of each Web page in terms of number of in-lined objects, and the size of each object are drawn at random from known distributions, as in [17]. Short, uniformly distributed user "think time" (2.5 s average) is used to simulate heavy Web usage. It would have been possible to generate the same traffic by adding more users to the simulation, a more taxing alternative on the simulator. When collecting download time samples, we use a small number of probe sessions (five for each of the HTTP versions) each with a different round trip time, which download fixed size pages (a 1 KB HTML index file with eight in-lined images of size 10 KB each) to eliminate the variations in download times due to different page sizes. Using fixed values allows us to more easily assess the performance obtained, without losing much of the applicability of the results. The file sizes and number of files per page for these users are close to median values found in recent Web traffic studies [26], which indicate that the complexity of Web pages has increased since earlier studies such as [24]. The aggregate traffic generated by the HTTP sources, when no other traffic is present (lossless network), amounts to about 33 Mbps.

### 2.2.2. Telnet

We model a Telnet client, as regulated by Nagle's algorithm. The client sends a 100 byte packet [2] to the server and waits for the acknowledgement (echo). The process is repeated after a random interval, such that the packet generation rate is approximately five characters per second, the rate for a fast typist [33]. The performance measure, *echo delay*, is the time it takes for a segment sent by the client to be acknowledged. The aggregate traffic generated by the Telnet sources, without other traffic (lossless network), amounts to less than 2 Mbps.

### 2.2.3. FTP

We use two types of FTP sources. The first, FTPlong, does infinite file transfers, the *throughput* of which is the performance measure of interest. The second, FTPshort, sends files with Pareto distributed files sizes (with shape parameter 1.2 and average 200 KB, the mean value of file transfers measured in an Internet backbone study [34]) separated by an exponentially distributed delay, with a 2 s mean, again to create heavy traffic. The performance measure for FTPshort sources is the file transfer time. When collecting transfer time samples, we use 10 probe sessions with different round trip times, which perform 200 KB fixed size transfers, in order to eliminate transfer time variations due to different file sizes. The traffic generated by

[2] This approximates the size of a typical Telnet packet containing a few characters, a 40+ byte TCP/IP header, as well as the MAC frame overhead. Since the latter is not present in *ns*, we include it here because the transmission time it adds on slow links may be perceptible to Telnet users.

the FTPshort sources is elastic, but cannot fully utilize a bottleneck larger than 100 Mbps by itself.

FTPshort sources are also used to create traffic on the reverse (ACK) path, i.e., from destination to source hosts. Such two-way traffic, is important because it is more realistic than one-way traffic, and involves interesting dynamics in the return queues, where the queueing and potential loss of ACKs can affect TCPs burstiness, and performance in general [36].

## 3. The effects of congestion on interactive applications

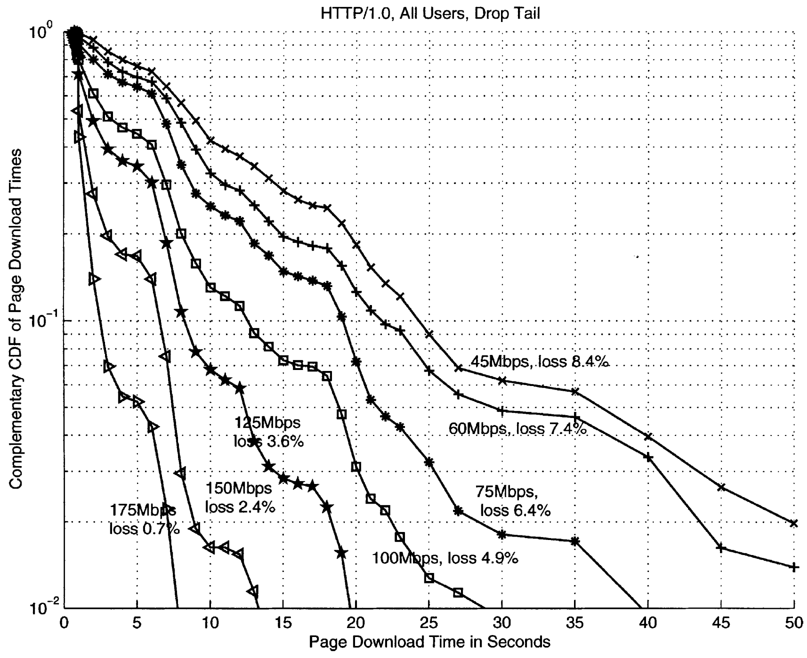
To motivate this study, we present in this section the results of simulations which illustrate the impact of congestion on the user-perceived performance of HTTP and Telnet.

The traffic scenario is as follows. Each source host has an active Telnet session, a Web client, and an FTPshort client at the corresponding destination host. Both HTTP implementations are considered, where one half the clients use HTTP/1.0 and the other half use HTTP/1.1. Fig. 2 shows the CCDF of page download times, that is, the fraction of downloads that exceed a certain time, assuming negligible server delays. Several curves are shown, corresponding to bottleneck link speeds ranging from 45 to 175 Mbps. The top figure shows the distributions for HTTP/1.0 download times experienced by the five probe sessions, which have different round trip times (ranging from 20 to 200 ms). Each curve is labeled with the average packet drop rate seen at the central link buffer. It is observed that, for all but the highest link speeds, a significant fraction of the downloads incur large delays. In addition, large variability can be seen in page download times for all link speeds. Other experiments we conducted have shown that, for a fixed total page size, the variability of HTTP/1.0 download times increases with the number of objects in the page. While we observed *goodput* figures approaching 100% in the experiments above, attesting to TCPs success in making good use of available network resources, the curves in Fig. 2 clearly indicate that the user-perceived performance of Web transfers is unsatisfactory.

Similar results are shown for HTTP/1.1 in the bottom figure. The first observation is that the delays incurred here are lower than those for HTTP/1.0. However, both the delays and variability are still larger than desired. Moreover, the use of different source servers for different objects within a page would reduce HTTP/1.1's performance benefits, as already pointed out in [23]. Note that the extent of HTTP/1.1's deployment is still limited, as observed in various measurement studies [2,23], which have found lack of deployment or compliance on both the client and server sides. In the rest of the paper, we focus on HTTP/1.0, noting that comparable results are obtained for HTTP/1.1.

Given the link speeds and the page size considered, expected download times are in the order of a few seconds. To explain the surprisingly large range of delays that are incurred, one might consider the different RTTs to be an important factor. However, this can be easily dismissed by looking at the CCDFs for individual probes (graphs not shown). While we find some small differences in the delay plots for the various RTTs, they all show the same spread in download times as in Fig. 2. Thus, the factor to be considered is the packet loss observed in the simulations, which ranges from about 8.5% for the 45 Mbps link to about 1% for the 175 Mbps link. Such drop rates are not uncommon in the Internet. For example, a measurement study of a large number of TCP connections at a busy Web server observed TCP segment loss rates in the Internet ranging from 5% to 7% [4]. However, the study does not show the resulting download delays. Here, we can show the packet drops' impact on the user-perceived performance of the Web transfers. In the experiments above, corresponding loss rates are observed for central link speeds of 100 and 60 Mbps respectively. As shown in Fig. 2, about 15% of HTTP/1.0 page downloads for the 100 Mbps central link (30% for the 60 Mbps link) incur delays larger than 10 s, the limit beyond which quality is typically perceived as low [7]. The percentage of downloads that exceed 10 s for HTTP/1.1 drops to 5% at 100 Mbps, and 20% at 60 Mbps.

The large delays and variability observed can be explained by examining the reaction of TCPs reliability and congestion control mechanisms to



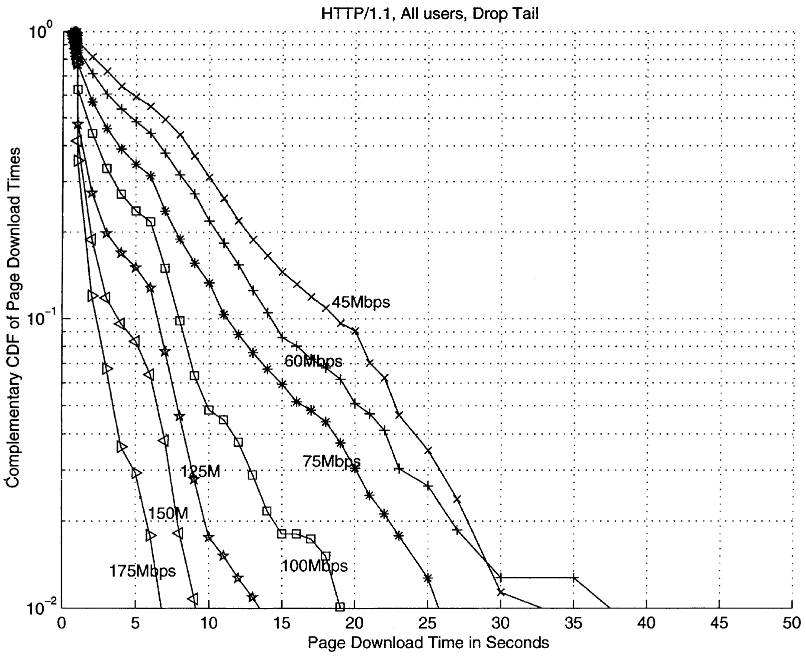


Fig. 2. CCDF of HTTP/1.0 and HTTP/1.1 downloads for different bottleneck speeds, and drop tail queues in all routers.

loss. First, the loss of connection establishment segments (SYN) is very costly to recover, given the large values commonly used for the initial retransmit timer (e.g., 3 or 6 s [10]). With the large number of short connections used in Web transfers, such loss is not a rare occurrence within a

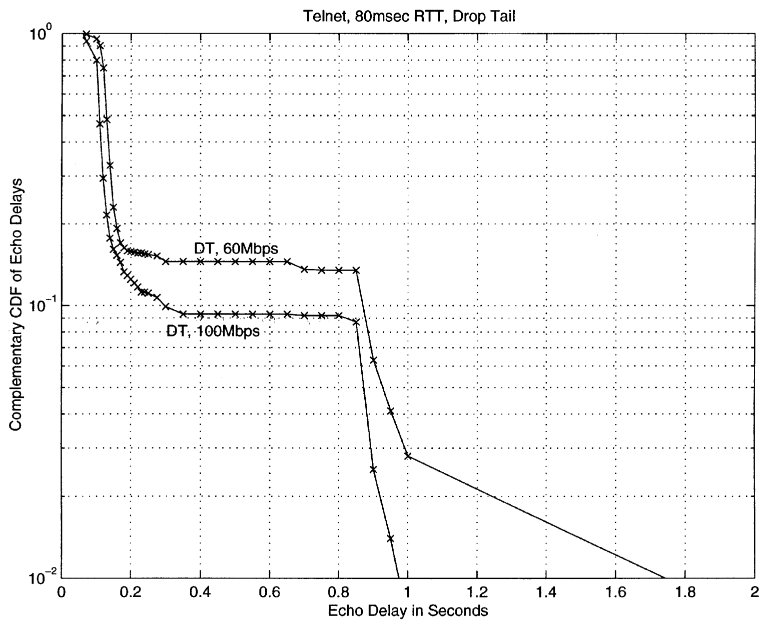


Fig. 3. CCDF of Telnet echo delays of an 80 ms RTT connection for 60 and 100 Mbps bottleneck links.

session. Second, TCPs loss recovery mechanisms are known to be inefficient when a connection's sending window size is small, as discussed in [16]. Indeed, for a small window, the number of duplicate ACKs received by the source is not sufficient to trigger the fast retransmit mechanism. Instead, TCP has to rely on the retransmit timer, typically resulting in a minimum idle time of 1 s. [3] Given that interactive transfers are usually short, they operate at small windows and are therefore particularly vulnerable to packet drops, as observed in [4]. In addition, the timeout is followed by slow start, where the connection operates at reduced rate. Finally, the "exponential retransmit backoff" rule typically doubles the retransmit timer value when a retransmitted packet is lost [31]. This means that the loss of successive retransmissions results in very large delays. Similar observations were made in a measurement study [5], where the causes of transaction delays are profiled by tracing TCP packets exchanged between Web clients and servers. The study shows the network is a significant component of total delay for medium sized transfers (Web objects), and packet loss is the main cause of response time variability.

Telnet is also very susceptible to loss, since it usually has only one packet in transit at a time. The loss of this packet always requires waiting for the retransmit timeout which, at 1 s minimum, introduces delays beyond the limit for good interactivity. In addition, successive losses would rapidly result in clearly unacceptable performance. For example, in the scenario described above, for the 100 Mbps link and the RTT range used, 1 in 10 echo delays takes about 1 s, while the others are received within acceptable delays, resulting in a bimodal delay distribution, as shown in Fig. 3. Significantly worse results are obtained for slower link speeds, as we show later in the paper.

These aspects of current TCP implementations show that they are not optimized for use in interactive applications. One may consider changing TCPs parameters to reduce the impact of large default values on performance. For example, the effects of reducing TCPs minimum timer value and the granularity of the timer are studied in [3]. Such modifications to TCP, as well as reducing the ini-

[3] The standard RFC for computing the retransmit timer places a 1 s minimum timer requirement, even when the actual timer computation results in a lower value [31].

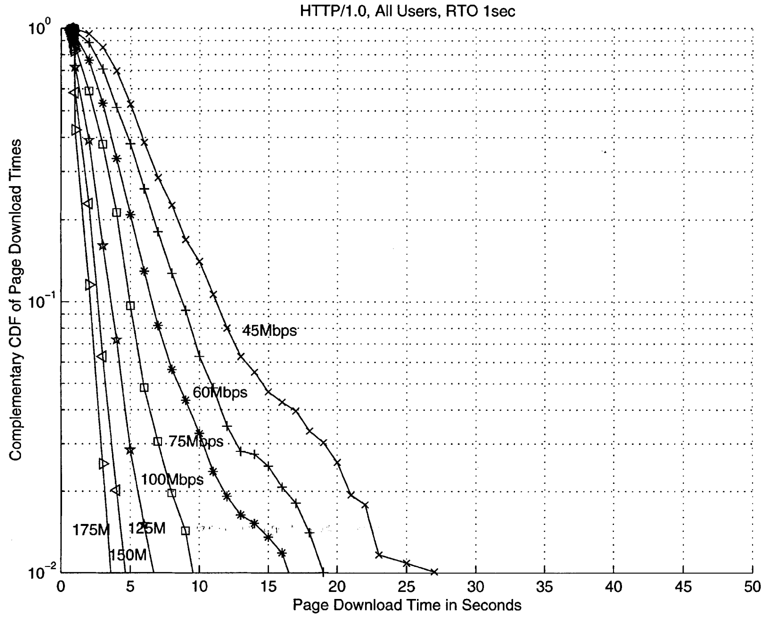


Fig. 4. CCDF of HTTP/1.0 downloads for initial RTO of 1 s.

tial retransmit timer value, might improve the performance of interactive applications by increasing their aggressiveness. For example, using a 1 s initial timeout (instead of the 6 s default used above) results in perceptibly lower HTTP/1.0 delays, as shown in Fig. 4. However, this value may result in performance degradation over long delay paths. In addition, concerns about the stability of the network may be raised as a result. Indeed, the loss rates observed in this scenario are about 20% higher than with "standard" TCP. Therefore, we do not further investigate such changes in this paper, and consider TCP implementations as currently deployed, and which follow the relevant standards for retransmission [10,31].

An alternative to modifying TCPs mechanisms, is to decrease the loss rate for interactive applications during congestion episodes, by giving priority to their traffic in the network. We explore this idea in the following sections.

## 4. QoS framework

In this section, we describe the network QoS mechanisms that are used in this study. We consider simple mechanisms, such as those introduced by the IETF DiffServ architecture [8], and the "Assured Forwarding" service in particular. We first review related work on the Assured Forwarding service, then we present the dropping functionality we use in routers. Lastly, we discuss the service agreements between the network and its clients, and the mechanisms needed for ensuring compliance with such agreements.

### 4.1. Assured forwarding

DiffServ enables service providers to give preferential treatment to some packets inside the network. A simple form of service differentiation within one queue can be provided by marking packets with multiple drop priorities, in association with a prioritized buffer management (dropping) mechanism. Such a service, called *Assured Forwarding*, was standardized in [22]. Four AF classes are defined, each with three drop precedence levels. The use of the AF service has been the subject of many studies, e.g., [12,20]. These studies focus on guaranteeing throughput for individual TCP connections, considering that an edge device, e.g., router, would mark users' traffic based on an agreed-upon profile. However, besides the need for appropriate provisioning along each connection's path (or some form of end-to-end admission control), this paradigm faces a number of challenges. First, in order to have control on individual connections' performance, the router needs to identify and keep track of all user connections. This might be prohibitive for short transfers associated with interactive applications. Second, it was found to be doubtful that TCP throughput can be controlled through such marking and dropping [29,32]. An alternative to this approach is to have sources pre-mark their own traffic. Previous work in this area has also focused on achieving an average rate for long transfers. Modifications to TCPs congestion control mechanisms, such as the use of two congestion windows or having different reactions depending on the marking of the lost packet, were required to obtain the desired performance [18,35]. In this study, we are mainly concerned with short transfers belonging to interactive applications, which require a fundamentally different type of service. These have not been addressed in previous AF-related studies.

### 4.2. Prioritized dropping

Following the AF specification, we consider queues where three packet priorities are supported, LOW, MED and HIGH. In this paper, we only consider TCP traffic. However, marked and policed UDP (e.g., layered video) traffic could have been mixed in the same queues, without affecting our results, or could have been mapped to a separate AF class. The integrated support of TCP and UDP applications in a network offering differentiated services is the subject of our current work.

The dropping function we use in network queues is a simple three-priority version of the random early detection (RED) dropping function [19]. Three average queue sizes are computed, one for each drop priority (*HIGH*$_q$, *MED*$_q$ and *LOW*$_q$), using the exponentially weighted moving average (EWMA) filter as in RED. When computing the queue size for a certain priority, packets that are at this priority level or higher are counted. We use

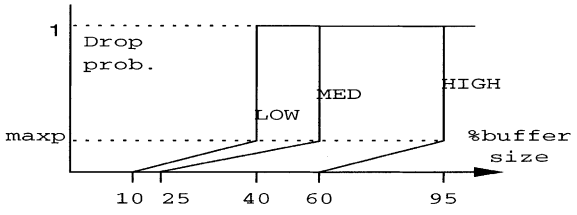


Fig. 5. Drop function used in network buffer management.

the same EWMA weight (e.g., 0.5), and *maxp* for the three levels (e.g., 0.1). The drop threshold values, as a percentage of buffer size are shown in Fig. 5. These settings have been validated through their use in numerous scenarios, spanning a large range of topologies, number of users, link speeds and traffic scenarios, where they consistently provided satisfactory performance.

### 4.3. SLAs and policing

To limit the aggregate rate of HIGH and MED priority packets in the network, service level agreements (SLAs) exist between the users and the network. We consider that SLAs specify per-user rate limits and allowable burst sizes for each of these two priorities, in the form of a token bucket profile. It is up to the users to pre-mark their traffic according to these contracts or to defer the marking to the service provider. On the other hand, it is the service providers' responsibility to ensure that SLAs are established in relation with the available network resources.

We *do not assume that any kind of trust must exist* between the network and the users. In order to police the marked traffic injected in the network, per-user mechanisms are present at the network edge. Policing actions may consist of dropping offending packets, or remarking them with a lower priority. Thus, the edge nodes effectively limit the aggregate rates of HIGH and MED priority packets that are admitted to the network. A key point we make is that, for scalability reasons, per-user agreements rather than per-connection agreements are made, i.e., the agreements cover the aggregate rate sent by the user which, at any one time, could be generated by only one or by many different connections. Given the policed rate agreements with the network, it is in the best interest of sources which mark their own traffic to implement *shaping* mechanisms that ensure conformance with the agreed-upon traffic profiles.

## 5. Application-based differentiation

In this section, we show the benefits of reducing the packet loss experienced by interactive TCP applications through giving priority to their traffic in the network. In addition, we discuss the limitations of prioritizing traffic strictly based on the application type, which motivate the need for a more flexible approach.

For simplicity, we use one AF class for TCP traffic. However, more than one class can be considered, allowing more flexibility in assigning applications to the different drop priorities. Using the three drop priorities available in an AF class, a natural mapping would be to send highly interactive applications' traffic, such as Telnet and network gaming, at highest priority (HIGH); interactive applications' traffic, such as Web, at medium priority (MED); and less interactive and more robust applications' traffic, such as FTP, at lowest priority (LOW). Thus, in the event of congestion, no Telnet packets would be lost, and the loss rate of HTTP packets would be limited.

The mechanisms required for such classification can be implemented in edge routers. Note that the router does not need to keep track of individual connections, since the marking could be determined on a per-packet basis (e.g., a simple scheme would use the well known port numbers). An obvious advantage of router-based marking is that no changes would be required in the stations. On the other hand, source-based mechanisms have the benefit of off-loading routers, and may also be the only possible option when using an *end-to-end IP layer encryption* scheme such as IPsec [6]. Indeed, IPsec hides all upper layer information beyond IP, and TCP and application-level information would only be available at the source.

In our simulator, we implemented the required mechanisms in the traffic sources, as shown in Fig. 6. When segments are released by TCP, the networking stack marks the appropriate field in the IP header based on the connection's application type. The sending of HIGH and MED priority packets is regulated using two token bucket shapers. Thus, for such packets to be transmitted by the source, sufficient tokens must be present in the corresponding token bucket shaper. This ensures that the marked traffic generated complies with the policer state at the router, which uses the same token bucket parameters to identify and drop offending packets, if any.

We repeat the experiment of Section 3, with this mapping of application traffic to priority levels.

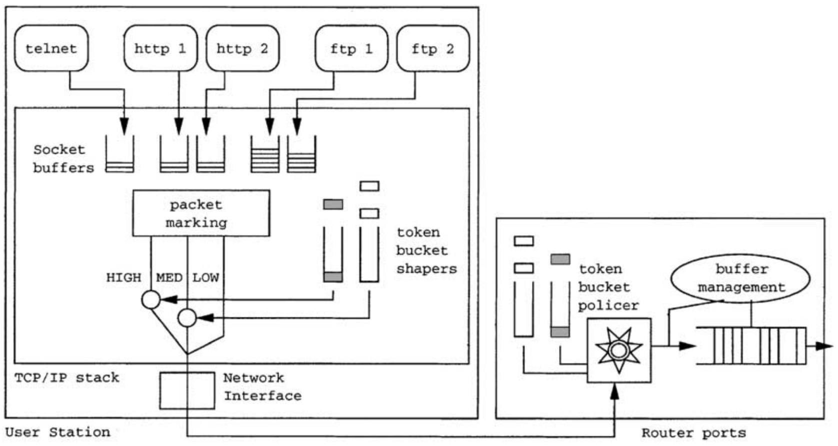


Fig. 6. Application-based differentiation mechanisms.

The aggregate high priority traffic in the reverse direction is chosen such that the link speeds studied range from under-provisioned to over-provisioned. In Fig. 7, we show the CCDF of HTTP/1.0 page download times, for 2 MED token bucket profiles (110 Kbps, 6000 bytes—dashed lines, and 250 Kbps, 6000 bytes—solid lines). These rates cover the interesting range of performance for the network conditions and the application requirements we are interested in. The token bucket profile for HIGH priority is large enough to minimize the delay of Telnet packets at the source (250 Kbps, 6000 B). As would be expected, download times are larger for the lower token rate, due to shaping delays at the source. Nevertheless, when the bottleneck link speed can accommodate the aggregate HIGH and MED traffic generated in both directions, the performance of HTTP is good for both profiles. Not shown is a similar plot for HTTP/1.1.

As would be expected, sending Telnet traffic at HIGH priority eliminates Telnet packet drops at the bottleneck link for all link speeds considered; thus, the distribution of echo delay does not exhibit the long delays seen in Fig. 3. On the other hand, as discussed earlier, Telnet is not only sensitive to delays from packet loss, but also to queueing delay. Multiple priority levels within one queue can be used to reduce packet drop rate, but not queueing delay. Therefore, if a Telnet connection's path goes over a low speed link, it may become necessary to use multiple queues, served by a weighted round robin (WRR) scheduler for example, to avoid long delays in a shared buffer. To illustrate this, we scale down by a factor of 10 all the link speeds in the topology, i.e., users are now connected to the network with 150 Kbps links and the bottleneck link speed is 10 Mbps. We use a traffic scenario comparable to the previous experiments. In Fig. 8, we show the CCDF of echo delays for a connection with 80 ms RTT, without differentiation (drop tail (DT) and RED), with application-based differentiation (APPL), and with multi-queue differentiation for different scheduler weights (lines labeled with the WRR scheduler weight of the Telnet queue). Although the shorter queue sizes associated with RED improve the packet delays compared to drop tail, it is clear that the quality obtained is poor for both, with large delays (several seconds) caused by packet loss and

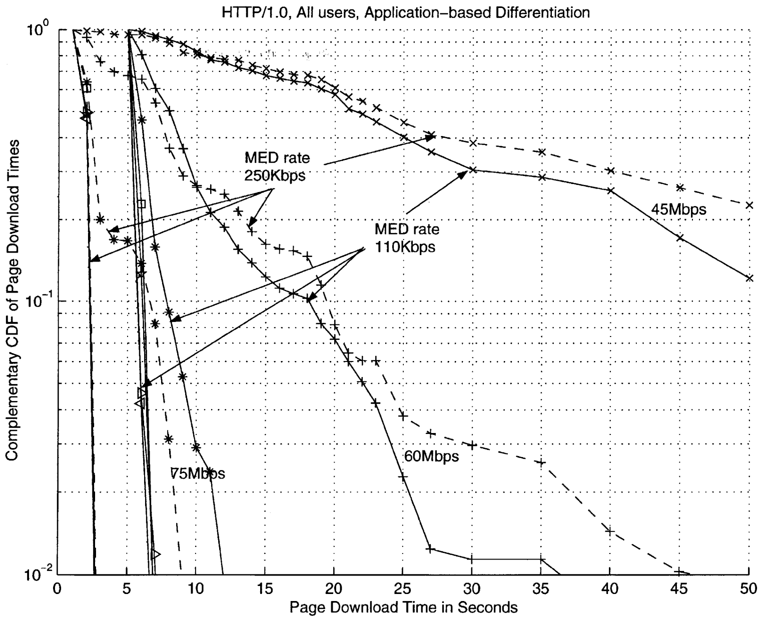


Fig. 7. CCDF of HTTP/1.0 downloads for different bottleneck speeds.

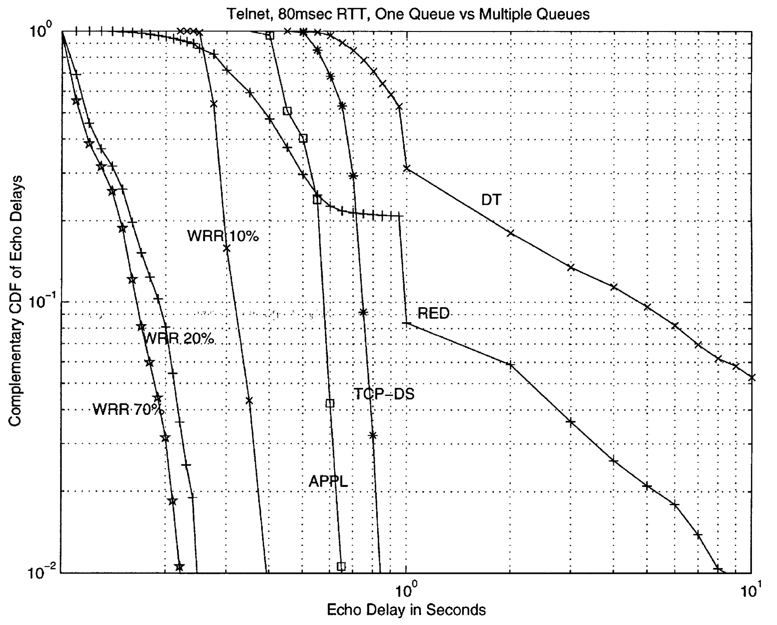


Fig. 8. Telnet echo delays for multi-queue and single queue differentiation.

retransmissions. The application-based prioritization provides significantly better performance, with all echos taking about 700 ms. However, the delays obtained are still larger than desired. Only with a separate queue, and with a large enough scheduler weight (e.g., 20% or more), can Telnet obtain the quality it requires.

The amount of Telnet traffic in the Internet is minimal (less than 1%, [34]), and giving it priority over other traffic would improve its performance without affecting other applications. However, this is not the case for all interactive applications, particularly the Web. The improvements in interactive applications' performance may therefore come at the cost of decreased performance for LOW priority traffic. In Fig. 9, we show how the LOW priority FTP traffic is penalized, for different MED token rates and a 75 Mbps bottleneck link. The plots show that the transfer times corresponding to the application-based differentiation are larger than for drop tail and RED, and increase with the MED token rate. For lower bottleneck speeds, where the aggregate of HIGH and MED traffic approaches the link's speed, the degradation in FTPs performance is significantly more severe. Nevertheless, the radical improvements in interactive applications' performance might justify the degradation in other applications' performance. Furthermore, as the plots for the different MED token rates indicate, the effects on low priority applications can be reduced if the contracted aggregate rates of higher priority traffic do not fully consume the network's resources. Unfortunately, in the DiffServ context, the lack of explicit resource reservation complicates network provisioning, and the likelihood of over-subscription on some links can be high.

Another limitation of this approach resides in the large variability among different sessions of one application type. For example, Web traffic (i.e., carried by HTTP) does not only consist of HTML code and small images for Web pages, or other interactive transfers. Indeed, measurement studies, such as [13], confirm what most Internet users know, that is, HTTP is also used to transfer large text documents and multimedia (audio and video) files. Without differentiating between HTTP sessions, interactive Web transfers may be affected by longer, less interactive ones. This can be addressed in several ways. If source-marking is

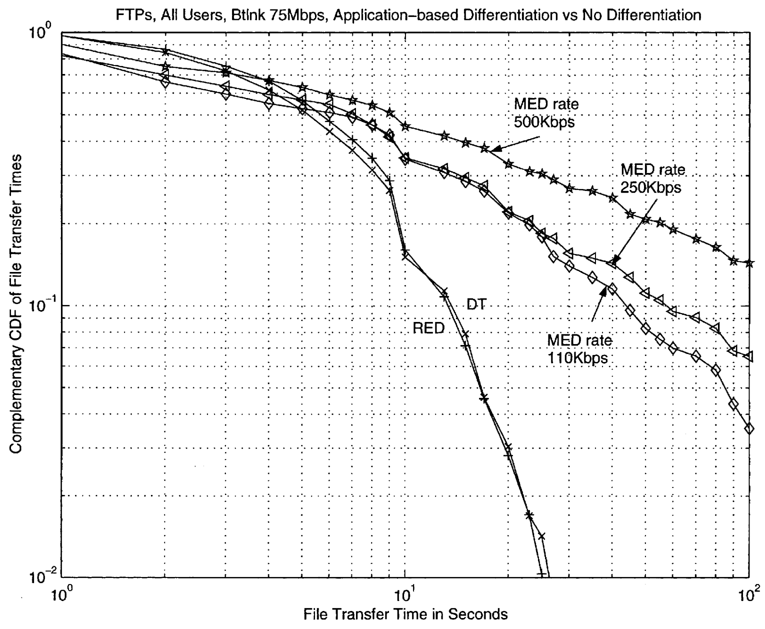


Fig. 9. CCDF of FTP file transfer times with and without application-based differentiation.

performed, a solution would be to assign transfers to the LOW priority class based on the content or transfer size. [4] However, this solution has the following drawbacks. First, the document size is not always available at connection setup time (e.g., for dynamically created content). Second, since some connections transfer different objects of different size and importance, as in HTTP/1.1, it might be necessary to modify the connections' priority during their lifetime. Finally, the selection of the appropriate size thresholds for mapping documents to the different priority levels may be difficult. Another option would be to add more levels of service (drop priorities) corresponding to the sub-categories within applications, e.g., by using several AF classes for TCP applications as mentioned earlier. Finally, it might be possible to achieve our goals without using more priorities, by assigning *individual packets* rather than entire connections to the different priority levels, as shown in the following section.

## 6. TCP-state based differentiation

The limitations in the application-based approach lead us to look for generic mechanisms, which can be used for any connection regardless of the application, and which would automatically prioritize short, interactive transfers while avoiding large negative impact on longer transfers associated with strict application-based prioritization. In this section, we show how the service differentiation available in one AF class can be used not only to achieve these goals, but also to improve the performance of non-interactive applications as well. We present mechanisms for two popular TCP versions (Reno and NewReno), which can be used, with minor modifications, for other versions.

Instead of mapping entire TCP connections to one drop priority, we propose here that the priority of each packet be determined individually. The mechanisms required at the sources of traffic are shown in Fig. 10. The main differences relative

[4] In this case, the network would be emulating the shortest remaining processing time scheduling studied in the context of HTTP servers in [14].

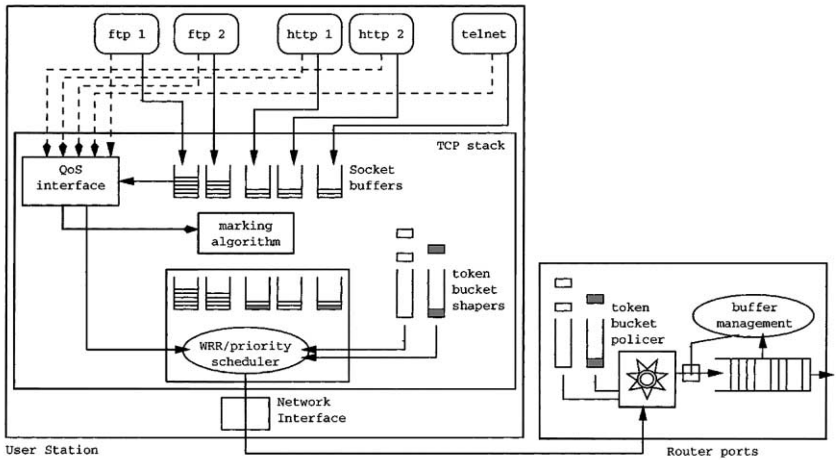


Fig. 10. TCP-state based service differentiation mechanisms.

to Fig. 6 are in the use of: (i) a TCP-state based marking algorithm, rather than application-type based marking, and (ii) an output link scheduler rather than simple token buffer shapers. In addition, an application programming interface (API) might provide the applications access to the settings of the marking and scheduling modules. An alternative would be to monitor the connections' performance and update the settings accordingly, or simply use default settings, based on each connection's application type for example. We do not go into further details concerning the API in this paper. The marking and scheduling mechanisms are described in more detail below.

## 6.1. Marking algorithm

A source host may have active connections to several different clients, going over widely different paths, and with correspondingly different performance. Given a *limited* budget of high priority tokens, it is important to carefully select the packets among the different connections to be marked as such. For example, consider a host with two active connections, the first going over a lightly loaded path, and the other going through a congested path. In this case, the first connection would not need high priority markings, while the second would significantly benefit from them. Therefore, in addition to taking into account the application the connection belongs to, the marking of individual packets for each connection should be based on the current state of the connection. Accordingly, the marking algorithm we describe here is based on TCP state and allows application-based differentiation to be performed through two control parameters.

### 6.1.1. The algorithm

Two basic premises are behind this algorithm. First, TCPs throughput is typically equal to the ratio of the *send window* size and the RTT, and therefore the window size is a good indication of the current performance of each connection. Hence, by prioritizing the connections based on their send window, a minimum level of performance can be guaranteed for each. Furthermore, the sending window of a connection that is performing well (i.e., going over an uncongested path) would be marked at low priority, freeing up high priority tokens, which can then be used to improve the performance of connections that need them. Nevertheless, if such a connection subsequently suffers packet drops, its window size will be reduced and it would automatically be marked at high priority. Second, as discussed in Section 3, the loss of some segments within a TCP connection has more impact than others on the performance of the connection. These segments are (i) the connection establishment segments, which are extremely important to the RTT sampling and the calibration of the retransmit timer, (ii) the segments sent when the connection has a small window, and (iii) the segments sent after a timeout or a fast retransmit. [5] The loss of such segments results in large idle time, as the connection waits for a retransmit timeout. Therefore, by sending them at HIGH priority, it is possible to improve TCPs resilience to congestion and packet loss. Therefore, we base the marking of packets on the size of the send window, in addition to identifying the other "special" packets and prioritizing them when necessary. [6] With this marking, the network during congestion can conceptually be seen as implementing a form of round robin service, where each connection is given a quantum of service in turn, allowing short connections (small jobs) to finish in predictable time.

A pseudo-code description of the algorithm is given in Algorithm 1. The *italicized* code corresponds to a randomized version which we describe below. The algorithm uses two window size thresholds, $HIGH_{\text{thresh}}$ and $MED_{\text{thresh}}$ to switch from HIGH to MED marking, and MED to LOW marking respectively. Basically, as the window increases and crosses the thresholds, packets are marked with decreasing priority. This means that a TCP connection has high priority as long as it is operating below a certain sending rate. Varying the setting of the thresholds allows fine grained control on the priority of a connection. Note that the sending rate of some applications is limited by nature (e.g., by human typing speed in Telnet) rather than by the TCP sending window. In this case, the window size does not reflect the actual sending rate. However, *congestion window validation* measures, such as proposed in [21], if implemented, would help address this issue and keep the marking aligned with the state of the connection. It is also possible to set the thresholds to mark such connections appropriately, as discussed in Section 6.1.2 below.

[5] For NewReno, we also prioritize segments sent during fast recovery.

[6] Typically, TCPs congestion window size is 1 segment when the SYN and timeout-retransmitted segment are sent. Therefore, the window-based marking would automatically prioritize these packets.

This marking algorithm, although based on TCP-state, requires no modification to the TCP mechanisms, and is applicable to all TCP versions, with minor modifications related to the internals of each version. As is clear from the pseudo-code, its addition to the TCP stack requires only a few lines of code. Since it does not change the congestion control mechanisms of TCP, the oscillations inherent to TCPs behavior are not eliminated. Instead, they are regulated, and the occurrence of extended idle times is minimized. As a result, when long-lived connections are examined at time scales relevant to humans (e.g., 2 or 4 s intervals) the performance is perceived to be steady, as shown in Section 6.3 below.

Notice that the window-based marking as described above abruptly switches between priorities as the window crosses thresholds. We have also experimented with a randomized variation that attempts to keep a fixed number of HIGH priority packets outstanding at all times (e.g., equal to the $HIGH_{\text{thresh}}$), with the goal of preventing sudden changes in performance. This is implemented through additional steps which mark packets with an appropriately chosen probability function. The additions, shown *italicized* in Algorithm 1, use $HIGH_{\text{thresh}}$ and $MED_{\text{thresh}}$ respectively as *approximate* limits on the number of HIGH and MED priority packets marked this way. A potential benefit could be an increase in the number of drops that are recovered through fast retransmit.

**Algorithm 1** [*Marking algorithm*].

```
if sendwnd $\leqslant HIGH_{\text{thresh}}$
        mark packet as HIGH
else if SYN or fast retransmit or fast recovery
        mark packet as HIGH
else if sendwnd $\leqslant MED_{\text{thresh}}$
        with probability $\frac{HIGH_{\text{thresh}}}{sendwnd}$
            mark packet as HIGH
        mark packet as MED
else
      with probability $\frac{HIGH_{\text{thresh}}}{sendwnd}$
          mark packet as HIGH
      with probability $\frac{MED_{\text{thresh}}}{sendwnd}$
          mark packet as MED
      mark packet as LOW
```

#### 6.1.2. Setting the marking thresholds

The marking thresholds provide control knobs that should be set according to the characteristics and requirements of applications. For Telnet, they are set at maximum window size in order for all packets to be sent at HIGH priority. Appropriately set thresholds automatically protect short transfers, such as most HTTP page downloads. Finally, they are set large enough to secure a minimum throughput for an FTP download.

With this marking, differentiation at finer granularities than application-level is possible, for instance, at the level of individual sessions of the same application. Thus, it is possible to prioritize a stock trading session, or a checkout page in an e-commerce site, over a regular "surfing" session, by giving them higher thresholds. Furthermore, differentiation between objects transferred using the same connection can also be done. For example, the user-perceived performance of Web browsing can be significantly improved by insuring that some components of a Web page, such as text and image bounding box information, are received within a few seconds, and used to generate an early layout of the page ("incremental loading") [7,9]. By using higher thresholds for these transfers, it is possible to guarantee a minimum level of quality for Web downloads. In a client-server context, the marking settings could be chosen by the server based on the connection's RTT, the application, the requested content, and/or the client (e.g., the user would "purchase" a certain service quality). Indeed, allowing users a choice of quality of service has been shown to increase user satisfaction and to optimize system usage [7].

### 6.2. Output link scheduler

As previously discussed, the marked aggregate has to be shaped at the source to comply with SLAs with the network. We describe in this section the mechanisms we designed and implemented for this purpose.

Since a potentially large number of connections could be active at a source host, packets from several connections may be queued waiting for tokens of a given priority. Furthermore, given that different connections may have different importance to the user, e.g., belong to different applications, it might be necessary to *prioritize* the allocation of high priority tokens to the different sessions. For this purpose, we implemented a hybrid scheduler/shaper module, which can be conceptually represented by the structure shown in Fig. 11.

The structure consist of a set of queues, one per connection, and a scheduler that services them. Having one queue per connection prevents sending packets from one connection out of order, and allows explicit control by the scheduler on the share of high priority tokens and output link resources received by each connection. The scheduler also

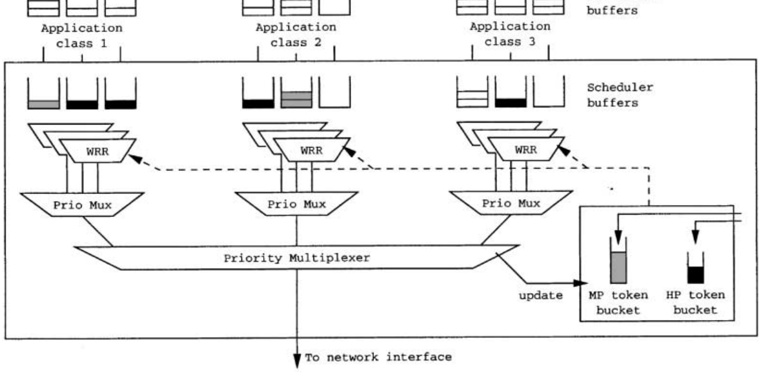


Fig. 11. Scheduler/shaper structure.

ensures that the aggregate traffic generated complies with the traffic profiles specified in the agreements with the network. For this study, where multiple application types are present at the sources, we make the following design choices.

We consider that the connections are organized into different classes, depending on the application they belong to. We define here three application classes, as in Section 5, in decreasing order of scheduling priority: highly interactive (Class 1), interactive (Class 2), and non-interactive (Class 3). We use the following service discipline. If any HIGH priority packet can be sent from a connection at Class 1, given the state of the shaper, it is dispatched to the network interface. Otherwise, MED priority packets, if any, are checked, followed by LOW priority packets. When all Class 1 connections are examined, and no packet can be sent, Class 2 connections are similarly checked, followed by Class 3 connections. These rules guarantee the lowest delay for the highest priority class, and derive from the classification we assume. If a packet is blocked waiting for a token of a certain marking priority at one class, no packet can be sent with the same marking priority at lower classes. This prevents a lower priority connection that uses a small packet size from starving higher priority connections. Within one class, connections are served in a weighted round-robin fashion, which, along with the marking thresholds, provides means for differentiation between connections belonging to the same class. In particular, connections marked with higher threshold potentially require a correspondingly larger share of the tokens, and the scheduler weights should be set accordingly. To avoid excessive blocking, the scheduler attempts to send the packet at the head of each queue at a lower priority (i.e., after remarking it) if this packet has been queued for a time longer than a certain threshold. [7] Since such packets are more likely to belong to low priority connections, this effectively means that these connections have a lower chance of getting admitted to the network during congestion than high priority connections. In terms of user-perceived performance, this user-oriented prioritized access to the network is superior to non-discriminating "implicit" admission control based on dropping new SYN packets during congestion (see for example [28]). Indeed, as argued in [11], such admission control mechanisms do not necessarily improve user-perceived performance, and may very well degrade it.

While the importance of regulating access to the link deceases as the link speed increases, the rate of *high priority tokens* could be *limited*, and regulating the access to this resource would prove beneficial, if not necessary. This scheduler therefore allows control on the differentiation between connections *at the source* as well as *in the network*. Note that other classifications, designs and scheduling rules are possible and could be more appropriate for different contexts. In particular, a simpler scheduler would probably be more adequate for a large server which handles only one application class, such as a Web server.

## 6.3. Results

We implemented the source marking and scheduling mechanisms described above in the simulator, and validated their performance through extensive simulations, for a variety of topologies and traffic scenarios. In this section, we present sample results which illustrate the performance improvements made possible. We show first that the TCP-state based approach provides similar improvements in interactive application performance to application-based differentiation. Then, we show that the performance of other applications (e.g., FTP) is not significantly affected. Finally, we present scenarios where the performance of such applications can be improved using the same mechanisms.

In the scenarios considered, the following settings were used for all HTTP and FTP connections: $HIGH_{\text{thresh}} = 4$ and $MED_{\text{thresh}} = 8$, and acknowledgments are marked with the priority of the data they correspond to. Since in these scenarios all connections are identically marked, equal weights within each application class are used in the

[7] In our simulations, we use 200 ms before downgrading from HIGH to MED, and another 200 ms for downgrading from MED to LOW.

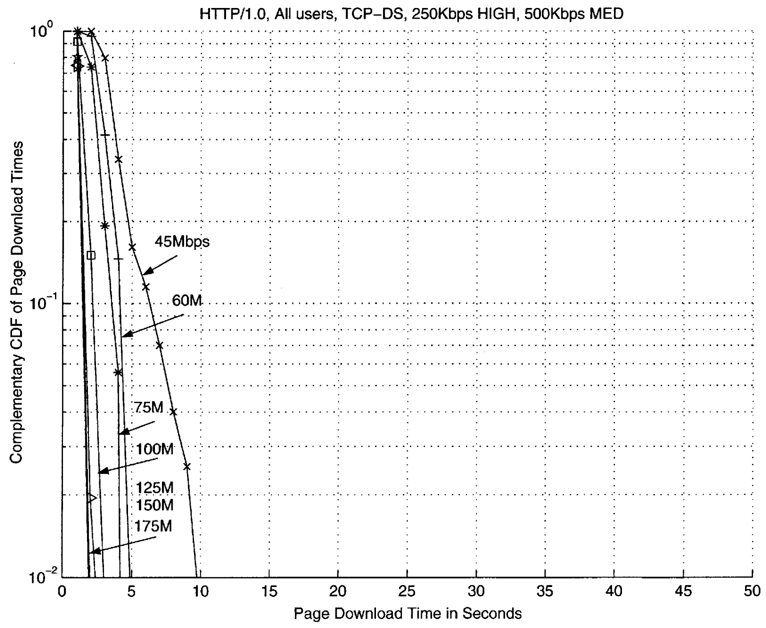
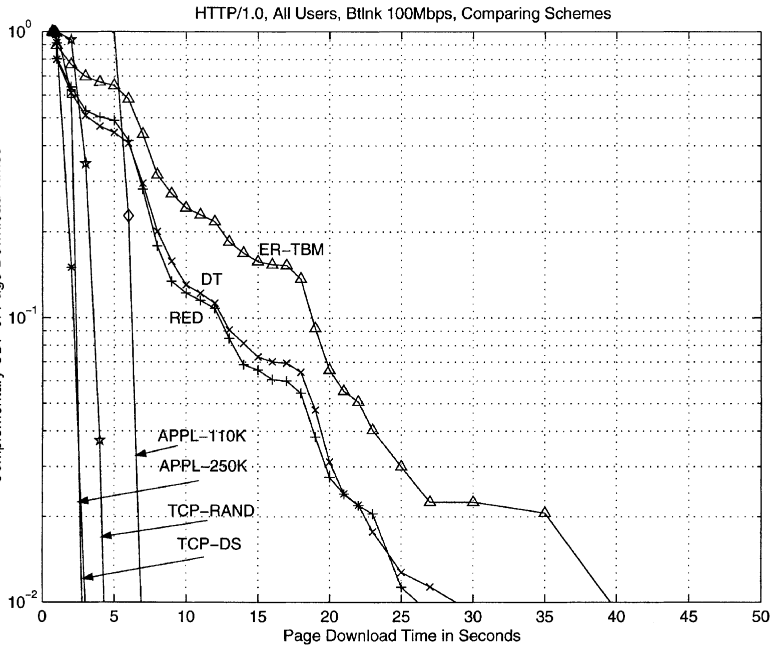


Fig. 12. CCDF of HTTP/1.0 downloads for different bottleneck speeds, HIGH rate 250 Kbps, MED rate 500 Kbps.

scheduler (therefore, a simple round robin scheduler would have been adequate). In Fig. 12 we plot the HTTP/1.0 CCDF for TCP-state based differentiation, for the same experiment as in Sections 3 and 5. In this case, as the bottleneck link is increased to 60 Mbps, all Web downloads complete within desirable delay limits, and with a high degree of predictability. In Fig. 8, the Telnet performance for this approach (TCP-DS), shows slightly more delay than application-based (800 vs 700 ms), due to generally higher queue occupancy.

A summary of performance results for interactive applications is shown in Fig. 13. In these figures we show the performance of drop tail, RED, application-based differentiation (for 250 and 110 K MED rates), TCP-state based differentiation (regular and randomized), and token bucket marking at edge routers (ER-TBM, 250 Kbps HIGH and 500 Kbps MED), for a 100 Mbps bottleneck link. The ideas that are illustrated here are the following. The applications' performance without service differentiation are comparable, whether drop tail or RED buffer management are used. In addition, marking of user *aggregate* traffic at the edge router with a token bucket marker, the "typical" approach for the AF service, does not result in adequate performance, even when all connections face the same network conditions as in the scenarios presented here, and may actually give worse performance than drop tail and RED. In general, different connections originating from the same source and going to different destinations may face different network conditions. In this case, a long transfer going over an uncongested path and performing significantly better than the other connections would receive most of the high priority markings at the router. This denies the benefits of differentiation to the connections that need it most. By explicitly selecting the packets to be prioritized, TCP-based differentiation provides good performance to interactive applications, similarly to application-based differentiation. Finally, the randomization in TCP marking does not have a large impact. Indeed, extensive simulations have shown that, although it results in better performance for HTTP/1.1 and file transfers in some cases, its effects are not quantifiable. Therefore, the use of the simpler algorithm is sufficient.

An advantage of the TCP-state based approach over application-based differentiation is



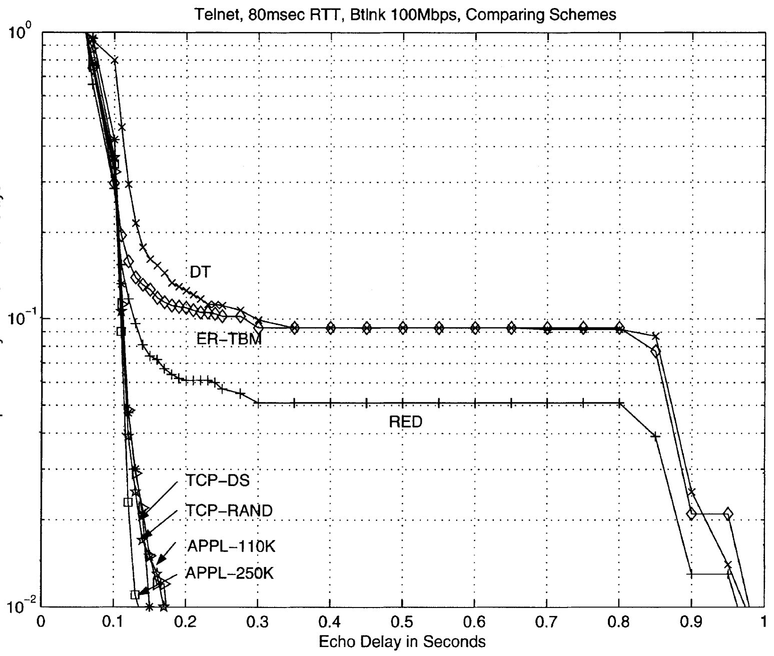


Fig. 13. CCDF of HTTP/1.0 downloads and Telnet echos for different schemes.

that lower priority applications are not heavily penalized. Overall, the CCDF of all FTP transfer times is very close to that of drop tail and RED (without differentiation). In addition, the performance is comparable in terms of the number of files transmitted per unit of time (see Fig. 14). In

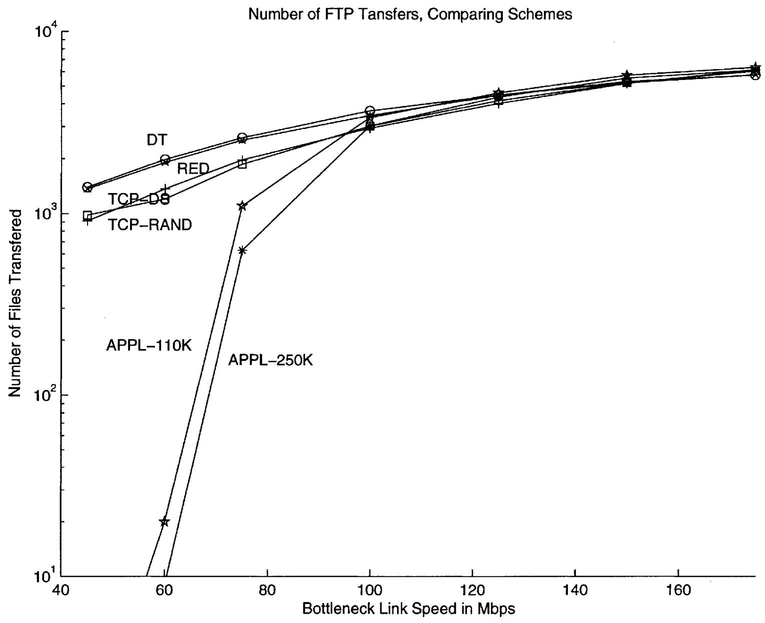


Fig. 14. Number of files transmitted by the probes for different schemes.

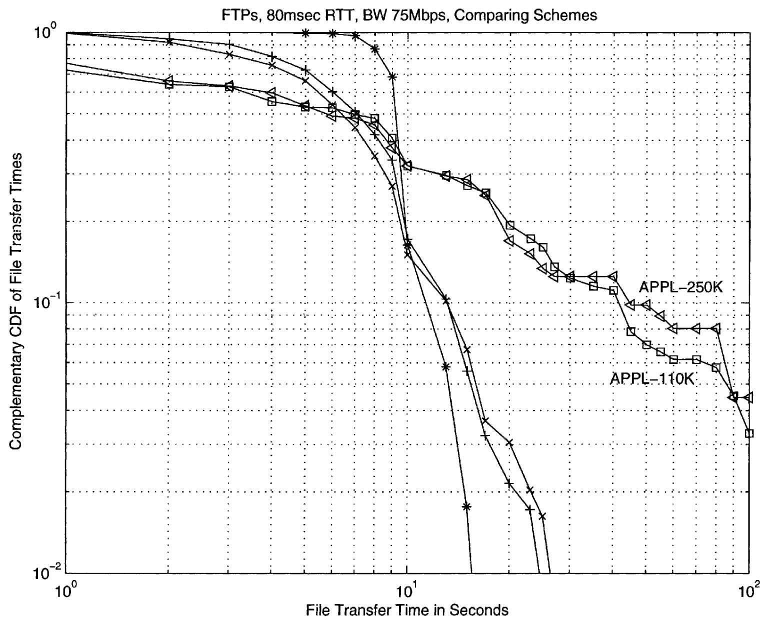


Fig. 15. CCDF of 200 KB file transfer times for different schemes.

contrast, the performance of FTP for the application-based approach at link speeds lower than 75 Mbps is very poor. Moreover, as can be seen in Fig. 15, the transfer times are made more predictable for individual users. These improvements are obtained because important FTP packets are prioritized as well. This means that users who are exclusively using such applications are not unduly penalized to the benefit of others.

In addition to improving interactive applications' performance, the TCP-based mechanisms can be used to improve the throughput of long FTP connections, and reduce the variations due to differences in RTT. To illustrate this, we consider scenarios where each source-destination pair performs one long FTP transfer (FTPlong). In Fig. 16, the average throughput of an FTP connection with a 200 ms RTT, is plotted against the HIGH priority threshold value used for its marking (the MED priority threshold is set at twice that value), for no differentiation (drop tail queues—DT) and TCP-state based differentiation (TCP-DS), and for 100 and 200 Mbps bottleneck link speeds. The threshold values for the other connections (with different RTTs) are fixed at values determined through similar experiments. Clearly, the throughput achieved for drop tail is independent of the marking algorithm settings. On the other hand, as the thresholds are increased, the connection's throughput reaches more than 85% of its fair share for TCP-DS, compared to less than 50% for drop tail, and stabilizes. The plots for the other connections (not shown here) indicate that they all achieve throughput close to their fair share at this stage. In other words, with similar network agreements, users with considerably different RTTs can independently achieve comparable throughput. Furthermore, the protection of sensitive packets results in smoother performance. Indeed, the variation coefficient (*std dev/mean*) of throughput samples for TCP-based differentiation is typically less than half that for regular (drop tail DT or RED) queues, as shown by the dotted lines in the figure. This translates into a relatively steady throughput during the lifetime of a connection, as apparent in Fig. 17, where the connection's throughput, sampled at 2 s intervals, is plotted for TCP-DS and DT over a 100 s period. Clearly, the throughput for TCP-DS has little variation around the mean, with no idle periods, in contrast to the one for DT. This would perhaps make it possible to use TCP for non-real time streaming applications, such as video on demand.

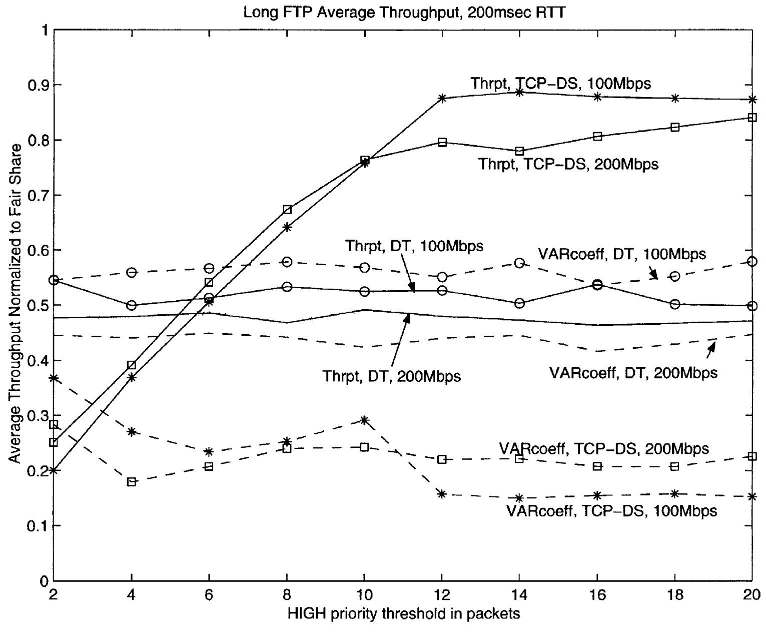


Fig. 16. Long FTP throughput normalized to fair share as a function of the marking thresholds.

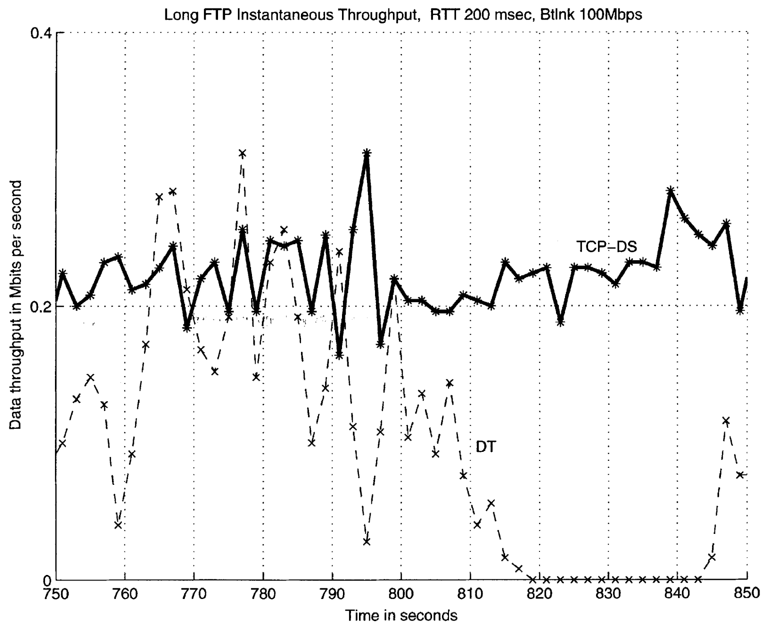


Fig. 17. Long FTP throughput vs time.

## 7. Conclusions

In this paper, we focus on congestion-induced delays in response times of interactive TCP applications. We show how these delays can be reduced using multiple service levels in the network, by giving preferential treatment to interactive applications' traffic in the network. We study an application-based and a TCP-state based approach to service differentiation, and describe the mechanisms required in the network and in traffic sources. Using simulations, with a large number of users and realistic traffic models, we show that both can achieve the goal of improving the performance of interactive TCP applications during network congestion episodes. Good user-perceived performance is obtained at times where severe degradation would have otherwise been experienced. In addition, by allowing other applications to use the high priority levels, the TCP-state based approach has the benefit of limiting the performance degradation they incur. Finally, we show how the performance of non-interactive applications can actually be improved using the same set of mechanisms.

## Acknowledgements

This work was supported in part by IBM through a Shared University Research equipment grant.

## References

[1] Network Simulator, ns version 2.1, available at http://www.isi.edu/nsnam/ns/.

[2] M. Allman, A web server's view of the transport layer, ACM Computer Communication Review (October 2000).

[3] M. Allman, V. Paxson, On Estimating end-to-end network path properties, in: Proceedings of SIGCOMM'99, August 1999.

[4] H. Balakrishnan et al., TCP behavior of a busy Internet server: analysis and improvements, in: Proceedings of IEEE INFOCOM, March 1998.

[5] P. Barford, M. Crovella, Critical path analysis of TCP transactions, IEEE Transactions on Networking 9 (3) (2001).

[6] Y. Bernet et al., A Framework for Integrated Services Operations over DiffServ Networks, Internet Draft, May 2000.

[7] N. Bhatti, A. Bouch, A.J. Kuchinsky, Integrating user-perceived quality into web server design, in: Proceedings of WWW'00, May 2000.

[8] S. Blake et al., An Architecture for Differentiated Services, RFC2475, December 1998.
[9] A. Bouch, M. Sasse, H.G. DeMeer, Of packets and people: a user-centered approach to quality of service, in: Proceedings of IWQoS'00, June 2000.
[10] R. Braden, Requirements for Internet Hosts—Communications Layers, RFC 1122, October 1989.
[11] J. Charzinski, Problems of elastic traffic admission control in an HTTP scenario, in: Proceedings of IWQoS'01, June 2001.
[12] D. Clark, W. Fang, Explicit allocation of best effort packet delivery service, IEEE Transactions on Networking 6 (4) (1998).
[13] M. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE/ACM Transactions on Networking (December 1997).
[14] M. Crovella, R. Frangioso, M. Harchol-Balter, Connection scheduling in web servers, in: Usenix Symposium on Internet Technologies and Systems, October 1999.
[15] L. Eggert, J. Heidemann, Application-level differentiated services for web servers, World Wide Web Journal 3 (1999).
[16] K. Fall, S. Floyd, Simulation-based Comparisons of Tahoe, Reno and SACK TCP, ACM Computer Communication Review (July 1996).
[17] A. Feldmann et al., Dynamics of IP traffic: a study of the role of variability and the impact of control, in: Proceedings of SIGCOMM'99, August 1999.
[18] W. Feng, D. Kandlur, D. Saha, K. Shin, Adaptive packet marking for providing differentiated services in the Internet, in: Proceedings of ICNP'98, October 1998.
[19] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993).
[20] M. Goyal, A. Durresi, R. Jain, C. Liu, Performance Analysis of Assured Forwarding, Internet Draft, February 2000.
[21] M. Handley, J. Padhye, S. Floyd, TCP Congestion Window Validation, RFC 2861, June 2000.
[22] J. Heinanen et al., Assured Forwarding PHB Group, RFC2597, June 1999.
[23] B. Krishnamurthy, C. Willis, Analyzing factors that influence end-to-end web performance, in: Proceedings of the Ninth International WWW Conference, May 2000.
[24] B. Mah, An empirical model of HTTP traffic, in: Proceedings of INFOCOM'97, 1997.
[25] S. Manley, M. Seltzer, Web facts and fantasy, in: Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.
[26] M. Mikhailov, C. Wills, Embedded Objects in Web Pages, WPI Technical Report, WPI-CS-TR-00-05, March 2000.
[27] J. Mogul, The case for persistent-connection HTTP, in: Proceedings of SIGCOMM'95, August 1995.
[28] R. Mortier et al., Implicit admission control, IEEE Journal on Selected Areas in Communications (12) (2000).
[29] B. Nandy, N. Seddigh, P. Pieda, DiffServ's assured forwarding PHB: what assurance does the customer have? in: Proceedings of NOSSDAV, July 1999.
[30] W. Noureddine, F. Tobagi, Improving the performance of interactive TCP applications using service differentiation, in: Proceedings of INFOCOM, June 2002.
[31] V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 2988, November 2000.
[32] S. Sahu, P. Nain, D. Towsley, C. Diot, V. Firoiu, On Achievable Service Differentiation with Token Bucket Marking for TCP, UMASS Technical Report 99-72.
[33] B. Shneiderman, Designing the User Interface, third ed., Addison-Wesley, Reading, MA, 1997.
[34] K. Thompson, G.J. Miller, R. Wilder, Wide-area internet traffic patterns and characteristics, IEEE Network (November/December 1997).
[35] I. Yeom, A.L.N. Reddy, Realizing throughput guarantees in a differentiated services network, in: Proceedings of ICMCS, June 1999.
[36] L. Zhang, S. Shenker, D. Clark, Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic, in: Proceedings of SIGCOMM, September 1991.

**Wael Noureddine** received the Bachelor of Engineering degree in Computer and Communications Engineering from the American University of Beirut in 1996, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, California, USA, in 1998 and 2002 respectively. His research interests include multimedia network applications, TCP applications performance and differentiated services.

**Fouad A. Tobagi** (M'77-SM'83-F'85) received the Engineering Degree from Ecole Centrale des Arts et Manufactures, Paris, France, in 1970 and the M.S. and Ph.D. degrees in Computer Science from the University of California, Los Angeles, in 1971 and 1974, respectively.

From 1974 to 1978, he was a Research Staff Project Manager with the ARPA project at the Computer Science Department, University of California, Los Angeles, and engaged in research in Packet Radio Networks, including protocol design, and analysis and measurements of packet radio networks. In June 1978, he joined the faculty of the School of Engineering at Stanford University, Stanford, California, where he is Professor of Electrical Engineering and Computer Science. In 1991, he co-founded Starlight Networks, Inc., a venture concerned with multimedia networking, and served as Chief Technical Officer until August 1998. His research interests have comprised packet radio and satellite networks, local area networks, fast packet switching, multimedia networking and networked video services, and multimedia applications. His current research interests include voice and video communication over the Internet, wireless and mobile networks, network design and provisioning, and network resource management.

Dr. Tobagi is a Fellow of the IEEE for his contributions in computer communications and local area networks. He is the winner of the 1981 Leonard G. Abraham Prize Paper Award in the field of Communications Systems for his paper "Multiaccess Protocols in Packet Communications Networks" and co-winner of the IEEE Communications Society 1984 Magazine Prize Paper Award for the paper "Packet Radio and Satellite Networks". He has served as Associate Editor for Computer Communications in the IEEE Transactions on Communications for the period 1984–1986, Editor for Packet Radio and Satellite Networks in the Journal of Telecommunications Networks for the period 1981–1985, Co-Editor of the special issue on Local Area Networks of the IEEE Journal on Selected Areas in Communications (November 1983), Co-Editor of Advances in Local Area Networks, a book in the series Frontiers in Communications (New York: IEEE Press), Co-Editor of the special issue on Packet Radio Networks of the Proceedings of the IEEE (January 1987), and Co-Editor of the special issue on Large Scale ATM Switching Systems for B-ISDN of the IEEE Journal on Selected Areas in Communications (October 1991). He has also served as Co-Editor of Advances in Local Area Networks, a book in the series Frontiers in Communications (New York: IEEE Press). He is currently serving as editor for a number of journals in High Speed Networks, wireless networks, multimedia and optical communications. He is a member of the Association for Computing Machinery and has served as an ACM national Lecturer for the period 1982–1983. He is co-recipient of the 1998 Kuwait Prize in the field of Applied Sciences.