Improving the Performance of Interactive TCP Applications Using Service Differentiation

Waël Noureddine, Fouad Tobagi

Abstract—Interactive TCP applications, such as Telnet and the Web, are particularly sensitive to network congestion. Indeed, congestion-induced queuing 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.

I. 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 the 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 msec for good user-perceived performance [30]. 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 sec [7]), and should have low variability to be 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], [30] 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, by prioritizing requests based on the application or the importance of the request for interactivity [14], or by using content replication and caching. In contrast, network delays, which form a significant part of total delay for Web transfers [4], [5], [24], [26], are usually outside the control of the provider or any other single organization, and are therefore 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 5 characters per second [30]. 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 queuing 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 queuing delays for individual packets. However, the delays due to TCP's 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 [26], 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 queuing 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

The authors are with the Department of Electrical Engineering, Stanford University, Stanford CA 94305. Email: {noureddine, tobagi}@stanford.edu

[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.

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

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 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. 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 II describes the simulation setup used in the study. Section III motivates the work, by illustrating the effects of congestion on Web page downloads and Telnet echoes. In Section IV, 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 V, we show how prioritizing interactive applications traffic in the network can improve their performance. Limitations in this approach lead us to look for a more flexible solution. In Section VI, 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 VII.

II. 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.

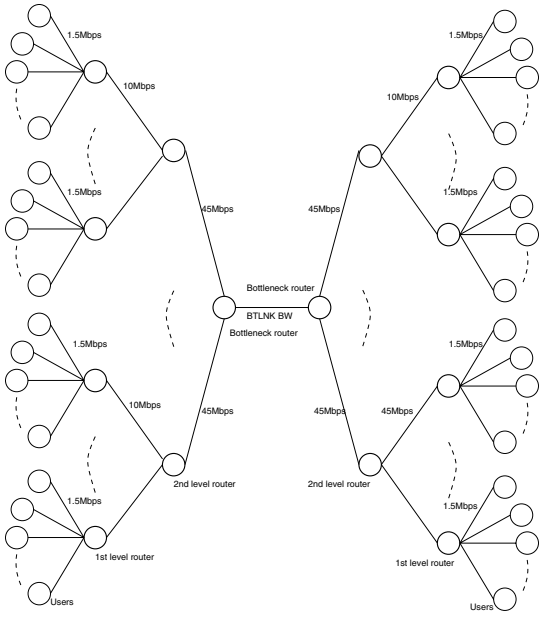

Fig. 1. Network Topology. Ten users are connected to each 1^{st} level router, 8 1^{st} level routers are connected to each 2^{nd} level router, and 5 2^{nd} level routers are connected to each bottleneck router.

A. 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, multi-hop tree topology, shown in Fig. 1. We use typical link speeds, starting from the Users side: 1.5Mbps (e.g. T1), 10Mbps (e.g., 10Mbps) and 45Mbps (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 communicating across the bottleneck. We have also experimented with different topologies, lower link speeds and fewer users, 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 intermediate nodes. Hence, the propagation delay of each link 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 200msec), thereby covering a 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 1.5Mbps, 10Mbps, 45Mbps and bottleneck links they are 64, 64, 250, and 500 packets, respectively.

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

B. 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 64KB (maximum unscaled value). We model traffic from the following representative TCP applications: Telnet, interactive Web, and FTP, generated in proportions that attempt to *roughly* approximate their real life counterparts, across the range of bottleneck links used.

Telnet. We model a Telnet client, as regulated by Nagle's algorithm. The client sends a 100 byte packet[2] and waits for the acknowledgment (echo). The process is repeated after a random interval, such that the packet generation rate is approximately 5 characters per sec, the rate for a fast typist [30]. The performance measure, *echo delay*, is the time it takes for a segment sent by the source to be acknowledged. The aggregate traffic generated by all the Telnet sources, without other traffic (lossless network), amounts to less than 2Mbps.

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 4 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 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 [16]. Short, uniformly distributed user "think time" (2.5 sec 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 (5 out of the 400) each with a different round trip time, which download fixed size pages (81KB, 1KB index file with 8 10KB images) to eliminate the variations in download times due to different page sizes, without losing much of the applicability of the results. The image sizes and number of images per page for these users are around median values found in a recent Web traffic study [25], which show that the complexity of Web pages has increased since earlier studies such as [23]. The aggregate traffic generated by the HTTP sources, when no other traffic is present (lossless network), amounts to about 33Mbps.

[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 because the transmission time it adds on slow links may be perceptible to Telnet users.

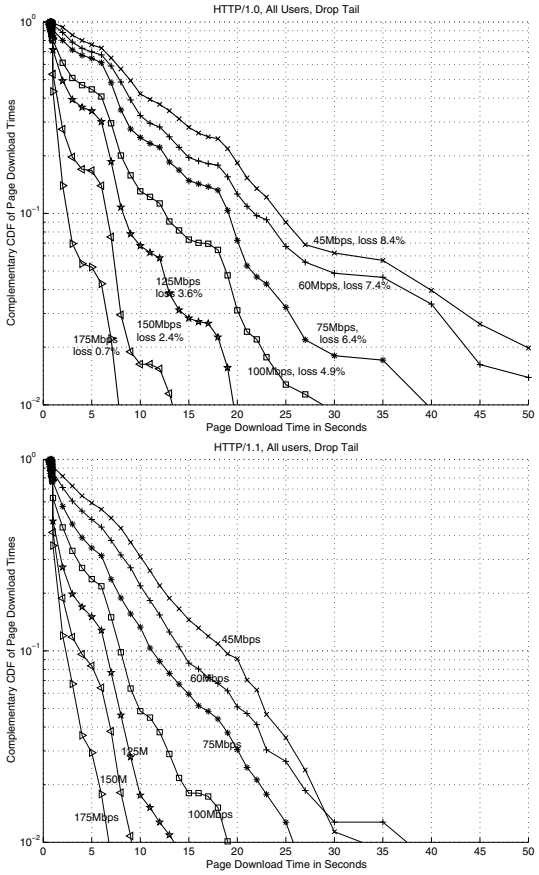

Fig. 2. CCDF of HTTP/1.0 and HTTP/1.1 downloads for different bottleneck speeds, and drop tail queues in all routers.

FTP. FTP sources send files with Pareto distributed files sizes (with shape parameter 1.2 and average 200KB, the mean value of file transfers measured in an Internet backbone study [31]) separated by an exponentially distributed delay, with a 2 second mean, again to create heavy traffic. The performance measure for FTP sources is the file transfer time. When collecting transfer time samples, we use 10 probe sessions with different round trip times, which perform 200KB fixed size transfers, in order to eliminate transfer time variations due to different file sizes. The traffic generated by the FTP sources is elastic, but cannot fully utilize a bottleneck larger than 100Mbps by itself.

FTP 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 queuing and potential loss of ACKs can affect TCP's burstiness, and performance in general [33].

III. The Effects of Congestion

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 FTP 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 45Mbps to 175Mbps. The top figure shows the distributions for HTTP/1.0 download times experienced by the 5 probe sessions, which have different round trip times (ranging from 20 to 200msec). 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.

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 [22]. Note that the extent of HTTP/1.1's deployment is still limited, as observed in various measurement studies [2], [22], 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 45Mbps link to about 1% for the 175Mbps 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 100Mbps and 60Mbps respectively. As shown in Fig. 2, about 15% of HTTP/1.0 page downloads for the 100Mbps central link (30% for the 60Mbps link) incur delays larger than 10sec, the limit beyond which quality is typically perceived as low [7]. The percentage of downloads that exceed 10sec for HTTP/1.1 drops to 5% at 100Mbps, and 20% at 60Mbps.

The large delays and variability observed can be explained by examining the reaction of TCP's reliability and congestion control mechanisms to 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 seconds [10]). With the large number of short connections used in Web transfers, such loss is not a rare occurrence within a session. Second, TCP's loss recovery mechanisms are known to be inefficient when a connection's sending window size is small, as discussed in [15]. Indeed, for a small window, the

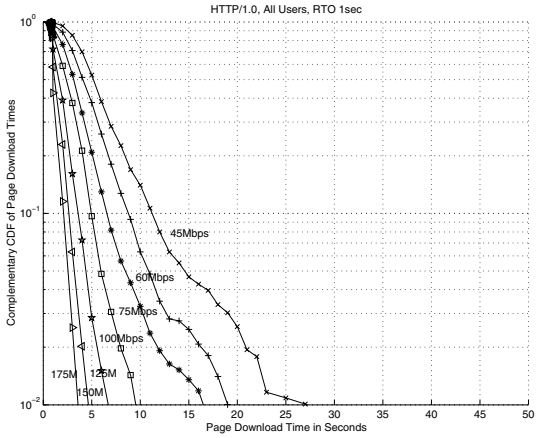

Fig. 3. CCDF of HTTP/1.0 downloads for initial RTO of 1sec.

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 second[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 [28]. 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 1sec 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 100Mbps link and the RTT range used, 1 in 10 echo delays takes about 1 second, while the others are received within acceptable delays, resulting in a bimodal delay distribution (not shown). 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 TCP's parameters to reduce the impact of large default values on performance. For example, the effects of reducing TCP's minimum timer value and the granularity of the timer are studied in [3]. Such modifications to TCP, as well as reducing the initial retransmit timer value, might improve the performance of interactive applications by increasing their aggressiveness. For example, using a 1 second initial timeout (instead of the 6 second default used above) results in perceptibly lower HTTP/1.0 delays, as shown in Fig. 3. However, this value

[3]The standard RFC for computing the retransmit timer places a 1 sec minimum timer requirement, even when the actual timer computation results in a lower value [28].

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

may result in performance degradation over long delay paths. In addition, concerns about the stability of the network may be raised as a result. 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], [28].

An alternative to modifying TCP's 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.

IV. 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 present the prioritized dropping function required in routers, then we discuss the service agreements between the network and its clients, and the associated mechanisms.

A. Prioritized Dropping

The *Assured Forwarding* service, standardized in [21], provides a simple form of differentiation within one queue with multiple drop priorities. Four AF classes are defined, each with 3 drop precedence levels. The use of the AF service has been the subject of many studies, e.g., [11], [19]. 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 [27], [29]. 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 TCP's congestion control mechanisms, such as the use of 2 congestion windows or having different reactions depending on the marking of the lost packet, were required to obtain the desired performance [17], [32]. 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.

Following the AF specification, we consider queues where 3 packet priorities are supported, LOW, MED and HIGH. In this paper, we only consider TCP traffic. However, marked and policed UDP traffic (e.g., layered video) could have been mixed in, without affecting our results, or could be mapped to a separate AF queue. 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 3-priority version of Random Early Detection (RED) [18].

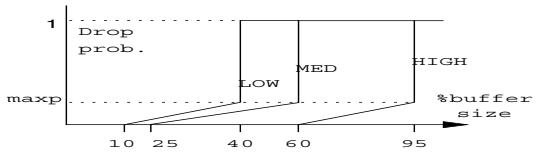

Fig. 4. Drop function used in network buffer management.

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 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. 4. 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.

B. 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 to the best interest of sources which mark their own traffic to implement *shaping* mechanisms that ensure conformance with the agreed-upon traffic profiles.

V. 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.

We first consider the use of one AF class for TCP traffic. Using the 3 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.

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

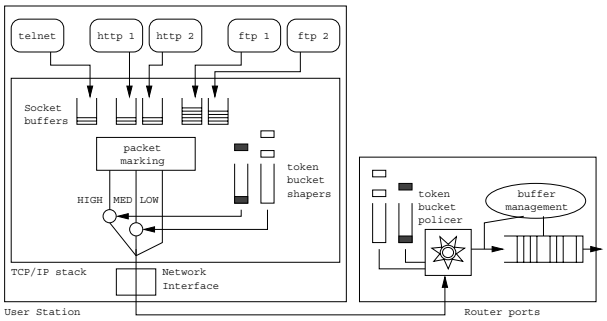

Fig. 5. Application-based differentiation mechanisms.

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. 5. 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.

We repeat the experiment of Section III, with this mapping of application traffic to priority levels. The aggregate marked traffic in the reverse direction is chosen such that the link speeds studied range from under-provisioned to over-provisioned. In Fig. 6, we show the CCDF of HTTP/1.0 page download times, for 2 MED token bucket profiles (110Kbps, 6,000 bytes - dashed lines, and 250Kbps, 6,000 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 (250Kbps, 6,000B). 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 packet drops for all link speeds, and the corresponding delays (results not shown). On the other hand, as discussed earlier, Telnet is not only sensitive to delays from packet loss, but also to queuing delay. Multiple priority levels within one queue can be used to reduce packet drop rate, but not queuing delay. Therefore, if a Telnet connection's path goes over a low

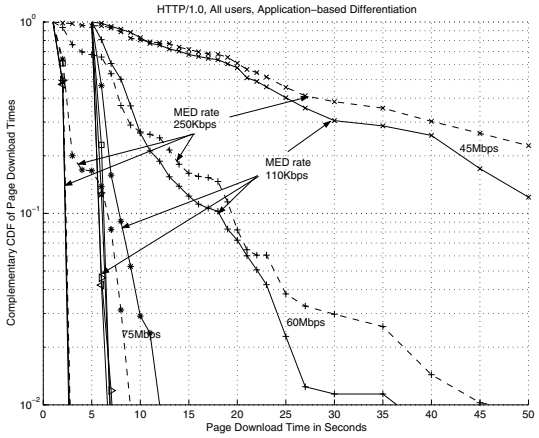

Fig. 6. CCDF of HTTP/1.0 downloads for different bottleneck speeds.

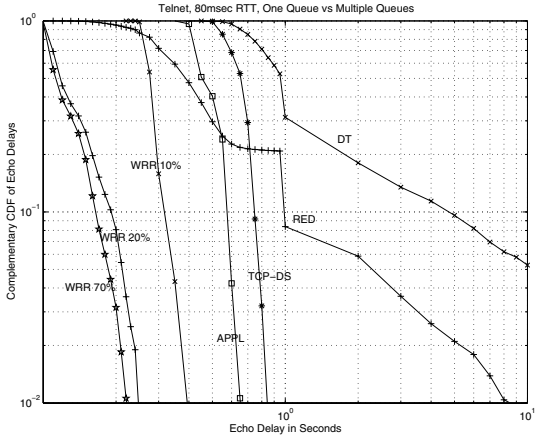

Fig. 7. Telnet echo delays for multi-queue and single queue differentiation.

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 150Kbps links and the bottleneck link speed is 10Mbps. We use a traffic scenario comparable to the previous experiments. In Fig. 7, we show the CCDF of echo delays for a connection with 80msec 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 retransmissions. The application-based prioritization provides significantly better performance, with all echos taking about 700msec. 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 improvements in Web and Telnet performance come at the cost of decreased performance for LOW priority traffic. In Fig. 8, we show how the LOW priority FTP traffic is penalized, for different MED token rates and a 75Mbps bottleneck link. The

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

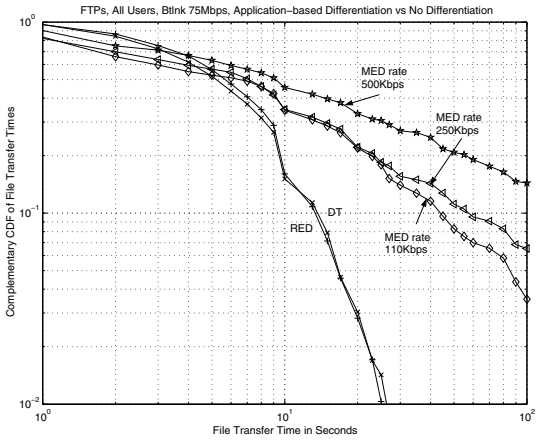

Fig. 8. CCDF of FTP file transfer times for application-based differentiation.

scenarios considered use 10 FTP probe sessions, with 200KB fixed file sizes, to isolate the variations in transfer times that are due to TCP's congestion control mechanisms. 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 approach the link's speed, the degradation in FTP's 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 limited if the contracted aggregate rates of higher priority traffic do not fully consume the network's resources. Unfortunately, in DiffServ, 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 [12], 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. If source-marking is performed, a solution would be to assign transfers to the LOW priority class based on the 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. 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 next section.

[4] In this case, the network would be emulating the Shortest Remaining Processing Time scheduling studied in the context of HTTP servers in [13].

VI. 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 to achieve these goals.

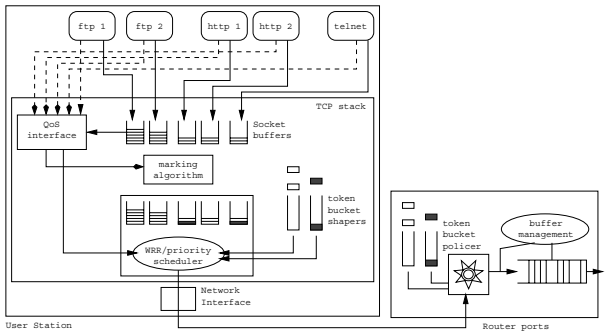

Fig. 9. TCP-state based service differentiation mechanisms.

Instead of mapping entire TCP connections to one drop priority, we propose here that the priority of each packet be determined individually. We describe a simple TCP-state based marking algorithm, and supporting mechanisms that are required at the sources of traffic: an application programming interface (API) and an output link scheduler (see Fig. 9). The API would provide the applications access to the settings of the marking and scheduling modules, or simply use default settings, based on each connection's application type. We do not go into further details concerning the API in this paper. The marking and scheduling mechanisms are described in more detail below.

A. 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. 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 TCP connection.

Two basic premises are behind the marking algorithm. First, TCP's 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 packets sent by a connection going over an uncongested path would be marked at low priority, freeing up high priority tokens, to be used for connections that need

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

them. Second, as discussed in Section III, 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 TCP's resilience to congestion and packet loss. Therefore, we base the marking of packets on the size of the send window, and we identify other "special" packets and prioritize 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 to finish in predictable time.

A pseudo-code description of the algorithm is given in Alg. 1. The *italicized* code corresponds to a randomized version which we describe below. The algorithm uses two window size thresholds, $HIGH_{thresh}$ and MED_{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.

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.[7] 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 TCP's behavior are not eliminated. Instead, they are regulated, and the occurrence of extended idle times is minimized. As a result, long-lived connections performance appears steady at time scales relevant to humans (e.g., 2 or 4 second intervals).

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 $HIGH_{thresh}$), with the goal of preventing sudden changes in performance. This is implemented through additional steps, shown *italicized* in Alg. 1, which mark packets with an appropriately chosen probability function ($HIGH_{thresh}$ and MED_{thresh} are *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.

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, and guarantee 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 (e.g., prioritize a stock trading session over a regular "surfing" session by using higher thresholds). 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, leading to increased user satisfaction and optimized system usage [7]).

Algorithm 1 Marking Algorithm.

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

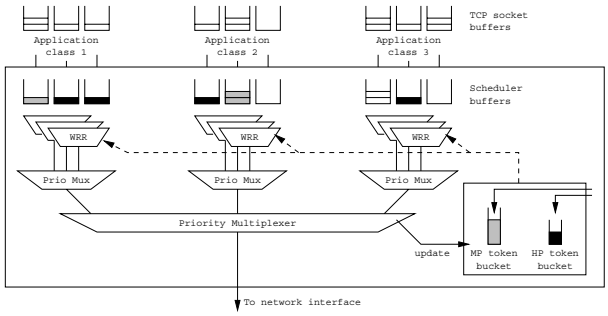

Fig. 10. Scheduler/shaper structure.

B. Output Link Scheduler

We describe in this section the mechanisms that we use for shaping the marked aggregate at the source to comply with SLAs with the network. For this purpose, we implemented a hybrid scheduler/shaper module, conceptually represented by the structure in Fig. 10, in order to *prioritize* the allocation of high priority tokens to the different active connections, according to connections' importance.

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 ensures that the aggregate

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

[6] Typically, TCP's 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.

[7] This version pertains to Reno and NewReno.

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 3 different classes, depending on the application they belong to, 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 ones. 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 larger share of the tokens, and the scheduler weights should be set accordingly. While the importance of regulating access to the link deceases as link speed increases, the rate of *high priority tokens* could be *limited*, and regulating the access to them would prove beneficial. 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 be more adequate for a large server which handles only one application, such as a Web server.

C. Results

The source marking and scheduling mechanisms described above were implemented in our simulator, and their performance has been validated 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.

In the scenarios considered, the following settings were used for all HTTP and FTP connections: $HIGH_{thresh} = 4$ and $MED_{thresh} = 8$, 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 scheduler. In Fig. 11 we plot the HTTP/1.0 CCDF for TCP-state based differentiation, for the same experiment as in Sections III and V. In this case, as the bottleneck link is increased to 60Mbps, all Web downloads complete within desirable delay limits, and with a high degree of predictability. In Fig. 7, the Telnet performance for this approach (TCP-DS), shows slightly more delays than application-based (800 vs 700msec), due to a higher queue occupancy.

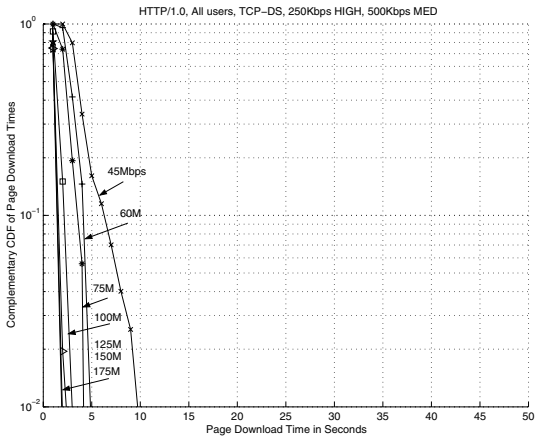

Fig. 11. CCDF of HTTP/1.0 for TCP-DS, HIGH 250Kbps, MED 500Kbps.

A summary of performance results for interactive applications is shown in Fig. 12. In these figures we show the performance of drop tail, RED, application-based differentiation (for 250K and 110K MED rates), TCP-state based differentiation (regular and randomized), and token bucket marking at edge routers (ER-TBM, 250Kbps HIGH and 500Kbps MED), for a 100Mbps 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 give adequate performance, even when all connections face the same network conditions, and may actually result in worse performance than drop tail and RED. In general, different connections originating from the same source and going to different destinations may experience vastly different conditions. Then, 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 at the source 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 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. 13). In contrast, the performance of FTP for the application-based approach at link speeds below 100Mbps is very poor. Moreover, individual CCDF plots show that the transfer times are made more predictable for individual users. These improvements are obtained because important FTP packets are

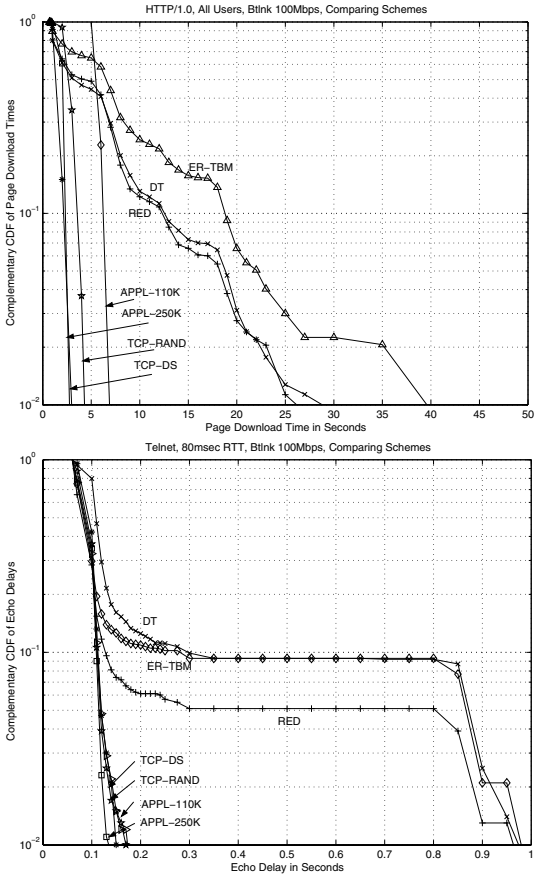

Fig. 12. CCDF of HTTP/1.0 downloads and Telnet echos for different schemes.

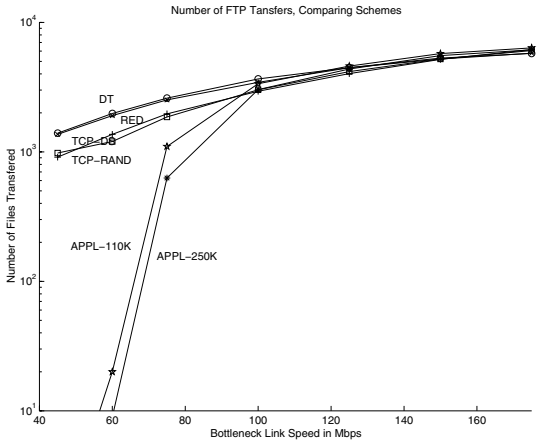

Fig. 13. Number of files transmitted by the probes for different schemes.

prioritized as well. This means that users who are exclusively using FTP are not penalized to the benefit of others.

For lack of space, we leave out results of simulations with long transfers, which show that the same mechanisms can be used to significantly reduce the variation in throughput achieved by long FTP connections with different round trip times. In addition, the throughput is considerably smoother than for drop tail and RED, as evidenced by a variation coefficient ($\frac{std\ dev}{mean}$) of throughput samples which is lower by half.

VII. 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.

References

[1] Network Simulator, ns version 2.1, available at http://www.isi.edu/nsnam/ns/
[2] Allman M., A Web Server's View of the Transport Layer, in ACM Computer Communication Review, Oct. 2000.
[3] Allman M., Paxson V., On Estimating End-to-End Network Path Properties, in Proceedings of SIGCOMM'99, Aug. 1999.
[4] Balakrishnan H. et al., TCP Behavior of a Busy Internet Server: Analysis and Improvements, in Proceedings of INFOCOM, Mar. 1998.
[5] Barford P., Crovella M., Critical Path Analysis of TCP Transactions, in IEEE Transactions on Networking, Jun. 2001.
[6] Bernet Y. et al., A Framework for Integrated Services Operations over DiffServ Networks, Internet Draft, May 2000.
[7] Bhatti N., Bouch A., Kuchinsky A.J., Integrating User-Perceived Quality into Web Server Design. presented WWW'00, Amsterdam, 2000.
[8] Blake S., et al., An Architecture for Differentiated Services, RFC2475, Dec. 1998.
[9] Bouch A., Sasse M., DeMeer H. G., Of Packets and People: A User-Centered Approach to Quality of Service, in Proceedings of IWQoS'00, June 2000.
[10] Braden R., Requirements for Internet Hosts, RFC 1122, Oct. 1989.
[11] Clark D., Fang W., Explicit Allocation of Best Effort Packet Delivery Service, in IEEE Transactions on Networking, Aug. 1998.
[12] Crovella M., Bestavros A., Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, in IEEE Transactions on Networking, Dec. 1997.
[13] Crovella M., et al., Connection Scheduling in Web Servers, in Usenix Symposium on Internet Technologies and Systems, Oct. 1999.
[14] Eggert L., Heidemann J., Application-Level Differentiated Services for Web Servers, in World Wide Web Journal, Volume 3, 1999.
[15] Fall K., Floyd S., Simulation-based Comparisons of Tahoe, Reno and SACK TCP, in ACM Computer Communication Review, Jul. 1996.
[16] Feldmann A. et al., Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control, in Proceedings of SIGCOMM'99, Aug. 1999.
[17] Feng W., Kandlur D., Saha D, Shin K., Adaptive Packet Marking for Providing Differentiated Services in the Internet, in Proceedings of ICNP '98, Oct. 1998.
[18] Floyd S., Jacobson V., Random Early Detection Gateways for Congestion Avoidance, in IEEE Transactions on Networking, Aug. 1993.
[19] Goyal M., et al., Performance Analysis of AF, Internet Draft, Feb. 2000.
[20] Handley M., et al., TCP Congestion Window Validation, RFC 2861, Jun. 2000.
[21] Heinanen J., et al, Assured Forwarding PHB Group, RFC2597, Jun. 1999.
[22] Krishnamurthy B., Willis C., Analyzing Factors That Influence End-to-End Web Performance, in Proceedings of the 9th International WWW Conference, May 2000.
[23] Mah B., An Empirical Model of HTTP Traffic, in Proceedings of INFOCOM'97.
[24] Manley S., Seltzer M., Web Facts and Fantasy, in Proceedings of the USENIX Symposium on Internet Technologies and Systems, Dec. 1997.
[25] Mikhailov M., Wills C., Embedded Objects in Web Pages, WPI Technical Report, WPI-CS-TR-00-05, Mar. 2000.
[26] Mogul J., The Case for Persistent-Connection HTTP, in Proceedings of SIGCOMM'95, Aug. 1995.
[27] Nandy B., et al., DiffServ's Assured Forwarding PHB: What Assurance does the Customer Have?, in Proceedings of NOSSDAV, Jul. 1999.
[28] Paxson V., et al., Computing TCP's Retransmission Timer, RFC 2988, Nov. 2000.
[29] Sahu S., et al., On Achievable Service Differentiation with Token Bucket Marking for TCP, UMASS Technical Report 99-72.
[30] Shneiderman B., Designing the User Interface, Addison-Wesley, 1992.
[31] Thompson K., Miller G. J., Wilder R., Wide-Area Internet Traffic Patterns and Characteristics, in IEEE Network, Nov./Dec. 1997
[32] Yeom I., et al.., Realizing Throughput Guarantees in a Differentiated Services Network, in Proceedings of ICMCS, Jun. 1999.
[33] Zhang L., et al., Observations on the Dynamics of a Congestion Control Algorithm, in Proceedings of SIGCOMM, Sept. 1991.

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.