Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Computer Networks 41 (2003) 779–794

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Preferential treatment for short flows to reduce web latency

Xuan Chen *, John Heidemann

Information Sciences Institute, University of Southern California, Suite 1001, 4676 Admiralty Way, Marina Del Rey, CA 90292-6695, USA

Received 17 January 2002; received in revised form 1 August 2002; accepted 19 November 2002

Abstract

In this paper, we propose short flow differentiating (SFD) algorithm to reduce the user-perceived web latency. This algorithm gives short flows preferential treatment. We implement SFD algorithm as a simple differentiated services policy and evaluate its performance in simulation. We find that SFD algorithm reduces the transmission latency of short flows and the response time to retrieve representative web pages by about 30%. Using web traces, we demonstrate that 99% web pages would be transferred faster. SFD penalizes long flows, but the penalty is well bounded. We further evaluate how different schemes trade-off the performance between short and long flows.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Short and long flows; Flow interaction; Web latency; Preferential treatment

1. Introduction

The Internet has a mix of traffic of all types including interactive traffic due to telnet and most web traffic; bulk, non-interactive traffic such as e-mail, some web traffic, and most file sharing traffic (FTP, Napster, etc.). The different needs of these traffic classes have been long recognized (for example with IP type-of-service bits [1,2]). However, recent approaches to traffic classification such as Differentiated Service (DiffServ) focus primarily on price—rather than application-based levels of service [3–5]. Further, the expectations of end-users should also be considered [6]. For example, end-users are likely to tolerate long delay while downloading movies, but easily feel upset if waiting too long to see Yahoo's web page.

A second observation about Internet traffic is that most *flows* are short, but most *bytes* are in long flows. Looking at web traffic, for example, recent measurements [7] show that 85% of all response flows sent from servers are less than 10K bytes, but they only account for about 20% of the total bytes transferred; the other 80% bytes are contributed by the longest 15% flows.

Finally, a common web design practice is to keep commonly viewed pages short to improve page view latencies for interactive browsing. This approach is widely discussed in web design books and has been shown to be consistent with heavy-tailed web file sizes [8]. For web traffic sent over

* Correpsonding author.
E-mail address: xuanc@isi.edu (X. Chen).

1389-1286/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
doi:10.1016/S1389-1286(02)00439-5

the HTTP/1.0 protocol [9], a new TCP connection is opened for every web object. This design approach suggests a strong correlation between short flows and interactive traffic (we discuss HTTP/1.1 [10] in Sections 3.2 and 4.4).

The implication of these three observations is that most interactive web traffic is in short flows and should be transferred faster. However, short flows suffer substantial queuing delay and packet loss due to bulk traffic in long flows. These observations suggest that interactive web performance could be improved by giving preferential treatment to short flows. This idea of "network hole plugging" (depicted in Fig. 1) was first suggested by John Doyle of Cal Tech—short flows should be able to quickly slip through the network, while long flows should "fill in the gaps" in short, bursty traffic.

Our paper proposes an algorithm to realize this goal and evaluates its effects through simulation. We use DiffServ to prioritize traffic, giving preference to short, presumably interactive flows. We implement the basic short flow differentiating (SFD) algorithm and its two variants, namely probabilistic and selective SFD algorithms, as a DiffServ policy in *ns-2* simulator. We evaluate the performance of SFD algorithm in simulations and find that SFD reduces the response transmission latency of short flows by more than 30%. This reduction increases as the network load becomes heavier. Through estimation, we find that SFD reduces 30% of the time to retrieve more than 90% web pages. We also study the sensitivity of our simulation results.

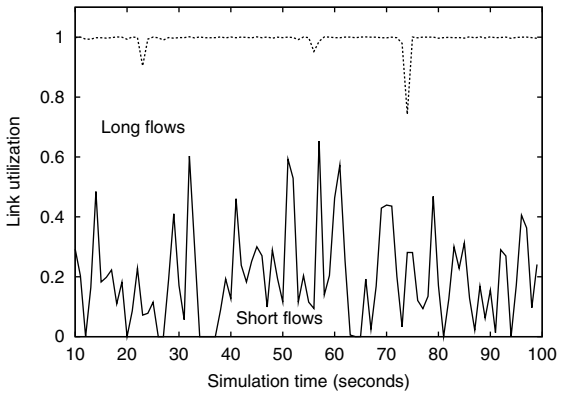


Fig. 1. "Network hole plugging": burst short flows leave holes for long flows.

There are two risks to this proposal. First, these improvements come at some penalty to long flows. We examine the affected flow size and assess this cost. We show that the penalty is bounded by SFD algorithm.

Second, the developing understanding of Internet stability is based on the importance of end-to-end congestion control [11]. This argument is based on the observation that the majority of Internet traffic is in congestion-controlled long TCP flows. Non-congestion controlled traffic and short TCP flows ("mice") are suspect because they are not TCP friendly and, in the aggregate, may threaten Internet stability. Although SFD favors short flows, its policies are selected so as not to destroy the benefits of end-to-end congestion control. We discuss these trade-offs in Section 3.3.

2. Background and related work

In this section, we present some background on web traffic measurement and modeling [7,12–16]. We also describe some related work on the DiffServ model and queuing management.

The NeTraMet Web Session Performance project at Caida [12] examined the composition of web traffic from network traces and found that about 75% of web traffic flows carry about 1K bytes or less. This result shows that the majority web traffic flows only consist of a few TCP packets.

Crovella and Bestavros [13] investigated the self-similarity in web traffic and discovered the size of web flows has a heavy-tailed distribution. This property is reproduced in a web workload generator SURGE [14] which is used to validate the web traffic model in our simulation [17].

The interaction between HTTP and TCP has also been studied [18–20]. Balakrishnan et al. [20] investigated the TCP behavior of a very busy web server (the web server for 1996 Atlantic Olympics) from its trace file. They explored the TCP behavior of both single and parallel connection(s) in terms of throughput and loss recovery and found that web traffic consists of many short TCP connec-

tions from a single host which show poor loss recovery performance. They enhanced TCP's loss recovery mechanism for improvement. We have similar observations in simulation and consider it as the main reason for the slow web transactions: large latency even for a small page. As opposite to modifying TCP implementation, we propose to apply SFD algorithm as a DiffServ policy to reduce the transmission latency of short flows.

The end-user perceived response time of web transactions (or web latency, we use these two terms interchangeably in this paper) is determined by many factors, such as web server load, network condition, and web page rendering time on browsers [15,21]. In this work, we mainly consider the circumstances where the web latency is network limited [21]. We also notice a complementary work which applies a similar idea on web server [22]. It implements a shortest-connection-first scheduling policy. Both work show smaller response time for short web connections and examine its cost: the unfairness or penalty suffered by long flows. We implement this scheduling policy and the traditional first-come-first-serve policy for web server and examine the combined effect of server and network scheduling policies on the end-to-end web latency.

The idea of SFD algorithm is quite similar to the shortest remaining processing time (SRPT) scheduling policy which minimizes the mean response time. Recent study [23] further shows that the "unfairness" of SRPT could be extremely small, especially when the processing time has a heavy-tailed distribution. Knowing the size of documents hosted, a web server can easily determine the processing time of a web request and apply SRPT scheduling policy to reduce the server's response latency. The shortest-connection-first scheduling policy (presented above) has been such an example. However, it is difficult to apply SRPT to networks because routers do not know flow size until the end of transmission. Our SFD algorithm solves this problem with a heuristic flow identification scheme as described in Section 3.2.

We implement SFD algorithms as a DiffServ policy. In DiffServ model (with assured services), given a profile that end users agree on, routers on the edge of networks (edge routers) keep flow states and mark the packets from flows obeying the profile as IN (in profile), otherwise as OUT (out of profile). Routers inside the network (core routers) give packets different dropping preference based on their marks (or codepoints): the OUT packets are more likely to be dropped than the IN packets when congestion happens.

By designing SFD under the framework of DiffServ, we eliminate the overhead for flow state keeping on core routers. This makes SFD easier to be deployed than packet scheduling mechanisms (fair queuing [24], for example), which can also be used to protect delay sensitive traffic such as telnet and short web flows.

Besides DiffServ, the Alternative Best-Effort Service (ABE) [25] is proposed to accommodate different requirement of delay-sensitive (green) and throughput-sensitive (blue) applications. Unlike ABE that requires applications to mark their packets as either green or blue and guarantees low latency for green packets as the expense of possible higher drop rate, SFD takes a different approach: routers classify traffic and protect short flows from packet loss by giving them preferential treatment.

RED with preferential dropping (RED-PD) [26] is an algorithm for routers to enforce fair share of bandwidth for flows. It detects high bandwidth flows and drops their packets preferentially. Although both RED-PD and SFD use the scheme of packet preferential dropping, SFD has a different goal from RED-PD: to improve web latency by protecting short flows from packet drops and large queuing delay.

Guo and Matta [27] conducted a very similar work simultaneously and independently. While showing slightly less performance improvement for short flows, they did not observe penalty to long flows, which we believe is possible under certain traffic pattern and network configurations. Besides, we evaluate SFD algorithm by estimating the response time to retrieve representative web pages and examine the combined effect of different server and network scheduling policies on the end-to-end web latency. We also study the performance of SFD algorithm in a broader range of traffic load and assess the possible effect on network congestion control.

3. The interaction among web traffic flows

In this section, we investigate the interactions between short and long flows by examining their transmission latency. The classification of short and long flows is based on *flow size* which is defined as the amount of data carried by a flow (1000 bytes, for example): *short flows* are those with flow size less than a certain threshold *th* and *long flows* otherwise. We propose the SFD algorithm to treat short flows preferentially.

We use *response transmission latency* to quantify the network effect in web latency. As shown in Fig. 2, the response transmission latency measures the time interval starting when a server sending out the first packet of a response and ending when the corresponding client receiving the last packet of the response. We focus on the dynamics in response transmission latency (or transmission latency, for short) in this work.

3.1. Interaction of short and long flows

To understand the interaction between short and long flows, we describe two simulations below. The detailed simulation methodology is described in Section 4.1. In these simulations, we configure the routers with droptail queues, which discard the incoming packets indistinguishably when the buffer is full. We measure transmission latency of short flows under different scenarios and show the mean transmission latency results in Fig. 3:

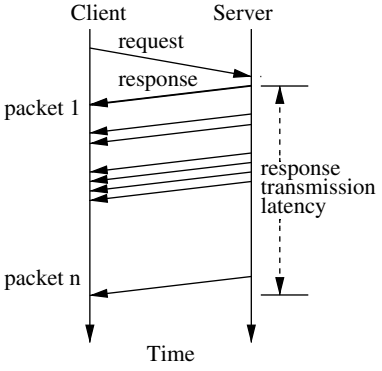


Fig. 2. Transmission latency.

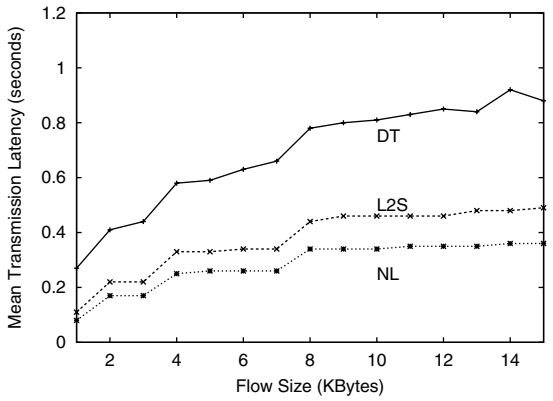


Fig. 3. Transmission latency of short flows with different traffic mix.

- *Flows with different sizes (denoted as DT)*. This is the baseline scenario where short flows compete with long ones.
- *No long flows (denoted as NL)*. We subtract long flows from web traffic. The effect of long flows is completely eliminated: traffic load is reduced as well as number of flows. The simulation result shows that the transmission latency of short flows is reduced by about 50% or more, which is the lower bound of short flows' performance.
- *Chop long flow into multiple short ones (L2S)*. We keep traffic load (the total amount of bytes transfered) unchanged, but intend to send them by short flows: an original long flow is chopped into several short ones (15K bytes or less). The simulation result shows that short flows still have about 30–50% smaller transmission latency compared to DT case.

These observations imply that the competition between short and long flows (for network resources, such as buffer space) slows down short flows' transmission. We present two reasons below:

- *Packet dropping*. Most routers deploying droptail queuing discipline discard packets indistinguishably under congestion. Because of the poor loss recovery performance [20], even a few packet drops can slow down short flows greatly. Further more, the retransmission of these dropped packets also consume network

resources (for example, bandwidth and buffer space) and make things even worse.
- *Queuing delay*. Even though routers may provide adequate buffer space to avoid packet dropping, short flows still suffer from the large queuing delay because they may be blocked by long flows which send tens of packets within one congestion window. Our simulations with infinite buffer space show that the transmission latency of short flows may increase especially when a very large web object (larger than 1M bytes, for example) is transferred.

Because of these two reasons, a dominant percentage of web transactions are slowed down. Some study at TCP level [20,27] also show similar observations. As a result, end users may experience a long waiting time even when fetching a simple web page with short text on it.

This leads to a natural question: *can we give preferential treatment to short flows to reduce their transmission latency?* We present such an algorithm below.

3.2. Differentiating short and long flows

We propose a simple algorithm to differentiate short and long flows, namely the SFD algorithm. We design this algorithm under the framework of DiffServ model: edge routers identify short and long flows and mark packets in short flows as IN and long flows as OUT; core routers give higher priority to IN packets so that they are protected against packet drops and large queuing delay.

Unlike the simple definitions described in Section 3.1, it is not easy for edge routers to determine flow size due to the lack of application level information and the dynamic contents hosted on web servers (for example, using CGI scripts to generate information for end-users).

Given the limited information carried by each packet, we propose a heuristic flow identification and packet marking scheme. Edge routers examine every packet header and record the amount of bytes having been sent by each flow so far as a flow state *f.bytes_sent*. They identify a flow as short and mark its packets as IN until its *f.bytes_sent* exceeds the flow identification threshold *th*.

In DiffServ model, only edge routers need to keep flow states; but edge routers do not need to keep flow state forever. We assume that edge routers time-out flow state after a short period p during which no traffic for that flow is observed. Not only does this minimize router state, but it also allows to treat long flows that are only intermittently used as separate short flows. For example, HTTP/1.1 traffic multiplexes multiple interactive transactions over a single TCP connection. With this timeout, portions of a long HTTP/1.1 flow separated by end-user think periods [7] would be treated as separate interactive short flows. In our work we set p to 1 s.

SFD applies RED [28] queuing discipline to handle IN and OUT packets. RED reduces queue length via early packet dropping: when the average queue length exceeds a minimum threshold (*min_th*), arriving packets are dropped with a probability roughly proportional to that connection's share of the bandwidth through the router; this probability is bounded by a maximum value (*max_p*) until the average queue length becomes greater than a maximum threshold (*max_th*) when all incoming packets are dropped.

The different treatment to IN and OUT packets can be realized by either preferentially dropping OUT packets or by a packet scheduling scheme (such as Fair Queuing [24]) with higher priority to IN packets. We choose the first approach in SFD. Specifically, SFD uses two *virtual* RED queues [29] for IN and OUT packets. Virtual queues assign different traffic classes different RED parameters and preserve packet order by actually putting incoming packets into one single FIFO queue. Each virtual queue handles the corresponding packets individually based on its virtual "queue length" and RED parameters. With stricter parameters, OUT packets are dropped preferentially when congestion happens. We also consider another option of using two separate RED queues with a priority packet scheduler. This approach imposes strict priority to IN and OUT packets and can not preserve packet order, which may cause starvation to long flows under certain circumstances such as applications intensively transmit data via short flows. We compare the architecture of both approaches in Fig. 4.

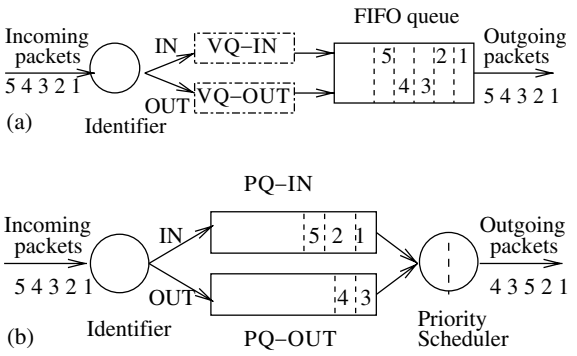


Fig. 4. Two approaches to give different treatment to IN and OUT packets: (a) virtual RED queue (VQ) with preferential dropping; (b) physical RED queue (PQ) with priority scheduling.

3.3. Concerns with SFD

A major concern in the design of SFD is that it does not destroy the benefits of end-to-end congestion control [11]. There are two risks here: first, it could so heavily favor short flows that long flows are starved; and second, even if long flows are not starved, the benefits of SFD could be so great that applications could intentionally chose to use multiple short flows instead of a single long flow.

To alleviate the first risk, we design two revised versions of SFD algorithm to avoid over-penalizing long flows The basic idea is to promote a fraction of packets in long flows to IN status. These algorithms are

- *Probabilistic SFD algorithm (denoted as P-SFD)*. Edge routers promote OUT packets with the probability as *th/f.bytes_sent*, which decreases as more packets being sent.
- *Selective SFD algorithm (denoted as S-SFD)*. Edge routers selectively promote a fraction of OUT packets, for example one of every K OUT packets. This modification has a tunable parameter K and requires edge routers to maintain a counter for each long flow.

Although both modifications "put" some OUT packets into IN virtual queue, the packet order is still preserved as we have discussed in Section 3.2.

SFD further avoids starvation of long flows through its choice of virtual RED queues. Although OUT packets suffer stricter RED parameters than IN (short-flow packets or promoted long-flow packets in P-SFD or S-SFD), all packets are placed in a single physical FIFO queue. Thus, once an OUT packet is accepted it will be sent. It is true that since short flows are given priority they will grow faster than they would in an undifferentiated network. However, the Internet community has previously granted some benefits to short flows, for example, by increasing the initial window size [30]. We suggest that SFD's policy of favoring the first *th* bytes of each flow is only a somewhat greater step toward favoring interactive traffic.

It is also possible that SFD might cause application writers to structure content or applications to send data in small pieces. Content is already structured in small pieces to reduce download time, thus some control over traffic is already outside the realm of protocol design. Fortunately, although some applications (for example, games) use custom protocols to get minimum latency, most major applications thus far are well behaved to make equitable (non-greedy) use of network resources. SFD should not change this trend.

4. Algorithm evaluation

We implement SFD algorithm as a DiffServ policy in *ns-2* [31] and evaluate its performance through simulations (denoted as SFD). We configure the two virtual RED queues under the guidance of a recent study [32]. We show the different RED parameters in Table 1.

We choose 15K byte as the flow identification threshold in our simulations ($th = 15$K bytes), which has been verified as a reasonable choice

Table 1
Parameters for the two virtual RED queues

Virtual RED queues	*max_th*	*min_th*	*max_p*	*w_q*
IN	30	10	0.02	0.02
OUT	24	8	0.10	0.02

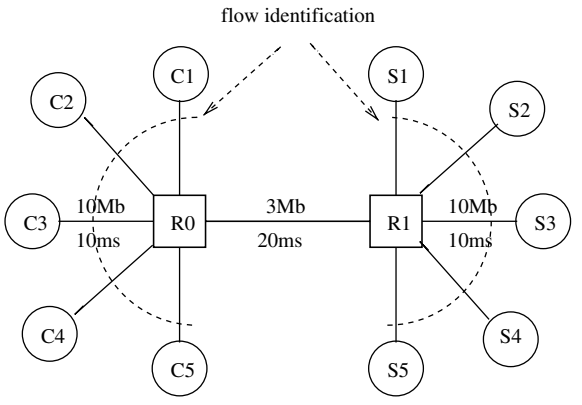


Fig. 5. Network topology in simulations.

from simulations (described in Section 4.3). With this threshold, 90% of the total traffic flows in our simulation are short which only contribute about 25% of the total network load; most of the network load are from long flows. This observation is consistent with the heavy-tailed distribution of web object size. [1] Similar statistic results are also observed in two real networks traces: *UCB96* collected by University of California, Berkeley in 1996 from the UC Berkeley dial-in IP modem bank [33] and *BU98* collected by Boston University in 1998 from a non-caching HTTP proxy server [16]. Both traces recorded the size of web objects requested.

4.1. Methodology

Fig. 5 shows the simulated network topology: 5 client hosts (C1–C5) and 5 web servers (S1–S5) are connected to router R0 and R1 by 10 Mbps links with 10 ms propagation delay; router R0 and R1 are connected by a bottleneck link with 3 Mbps bandwidth and 20 ms propagation delay. Routers R0 and R1 identify short and long flows. We relax this topology in Section 4.6.

This topology can be viewed as a simplified version of the interconnection between two ISPs corresponding to R0 and R1, where one provides user access (R0) and the other provides WWW services (R1). With this specific topology requests (from clients to servers) and responses (from servers to clients) are separated in two directions. Requests (usually contain one small packet, 48 bytes in our simulations) are transmitted under very light traffic load and will not increase the overall web latency.

In the web traffic model in *ns-2* [17], clients initiate a series of web sessions, each retrieving some web pages from randomly chosen servers. A web page consist of several web objects, which should be modeled as a heavy-tailed distribution in size [13] to produce the self-similarity in web traffic. We model these attributes with different probabilistic distributions according to a latest study on web traffic measurement [7] (as summarized in Table 2).

The network load can be adjusted by varying the number of web sessions and the mean inter-arrival time between sessions (i.e. load parameters). We study three network load conditions in our simulation: light load, medium load, and heavy load. The corresponding load parameters, utilization and packet loss rate of the bottleneck link are shown in Table 3. Unless otherwise specified, we conduct simulations under medium network load, which we believe is of most interest.

We deploy SFD algorithm to all routers and compare the simulation results with the DT scenario described in Section 3.1. Since SFD algorithm uses virtual RED queues, we also evaluate the effect of applying RED (denoted as RED). We deploy RED queues to all routers and configure them with the parameters for OUT packets to limit the number of packets in buffer. We expect smaller transmission latency for short flows because the queuing delay is reduced and packets from short flows are unlikely to be dropped because the relatively small amount of bandwidth they consume.

We run the simulations for 12,000 seconds and record data after a warming up period of 2000 seconds to avoid transient effects. We measure transmission latency for flows with different sizes. and present its mean and standard deviation as a function of flow size. For accuracy, we aggregate these results for very long flows. We do not show the confidence interval with data in the graphs

[1] Given the same threshold, traffic flows generated by a web traffic model with log-normal distributed object size is consist of 99% short flows.

Table 2
The attributes of web model and corresponding distributions

Web model elements	Element attributes	Probabilistic distributions	Parameters
Web session	Number of web sessions	Constant	Value: 1250
	Time interval between sessions (s)	Exponential	Mean: 15
	Number of web pages per session	Exponential	Mean: 100
Web page	Time interval between pages (s)	Exponential	Mean: 10
	Number of web objects per page	Exponential	Mean: 3
Web object	Time interval between web objects (s)	Exponential	Mean: 0.01
	Object size (K bytes)	Pareto II	Mean: 12, Shape: 1.2

Table 3
Different network load

Network load	Session number	Inter-session time (s)	Link utilization (%)	Loss rate (%)
Light	1000	25	30	0.1
Medium	1250	15	40	1
Heavy	1400	9	70	10

because it is very small: 1–2% compared to the data (with 95% confidence level).

4.2. Performance improvement of short flows

Fig 6(a) and (b) show that the mean and standard deviation of transmission latency for short flows under different scenarios (DT, RED, and SFD). It is clear that SFD algorithm gives much better performance. In Table 4 (scenario: simple), we quantify this improvement as the reduction in transmission latency for short flows.

We observe that SFD algorithm reduces 32% of the mean transmission latency for short flows on average. SFD also shows much smaller variation in transmission latency, the standard deviation is only half of that in DT case, which indicates that the transmission of short flows are more predictable with SFD. Given the percentage of short flows, more than 90% web flows show better performance.

Recalling our analysis in Section 3.1, it is interesting to note that the transmission latency with SFD is quite similar to that of L2S case. Given the lower bound of transmission latency we could expect (as shown in NL case), SFD algorithm achieves pretty good performance (as of 60% of the best case).

On the other hand, RED does not show better results than droptail queue. Similar result has also been observed in a recent study [32], which shows that RED does not outperform droptail queues for Web traffic under most traffic load.

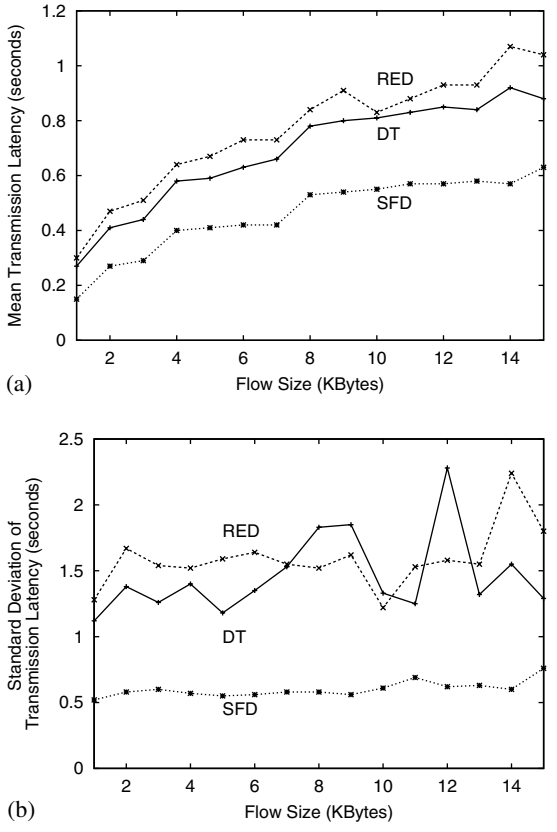


Fig. 6. The transmission latency of short flows under different scenarios: (a) mean transmission latency; (b) standard deviation of transmission latency.

Table 4
SFD reduces the transmission latency for short flows under different scenarios (compared with DT, negative percentage means increase)

Scenarios	Reduction in transmission latency		
	Max (%)	Min (%)	Mean (%)
Simple	42	27	32
2-Tier	26	20	23

4.3. Impact on long flows

SFD algorithm reduces transmission latency of short flows at the cost of penalizing long flows. We quantify this penalty and the affected flow size in this section. We show that the penalty of long flows is bounded by the design and implementation of SFD algorithm. With the two virtual RED queues and the corresponding RED parameters, long flows are given the buffer space for about 15 packets which corresponds to about 30% of the bottleneck link capacity. Therefore, the transmission latency for long flows will not be increased by more than two times.

In Fig. 7, we show that the mean and standard deviation of transmission latency for flows with size up to 1M bytes. Fig. 7(a) depicts that SFD also reduces transmission latency of some long flows (up to about 40K bytes). This result is not unexpected because the first 15K bytes are always marked as IN (as described in Section 3.2) and protected by SFD upon congestion. Since the first a few packets in long flows are as fragile to loss as short flow packets, SFD actually protects these long flows, which is especially important for those long flows up to about 30K bytes because at least 50% of their packets are within this range.

However, we do observe that very long flows suffer from SFD algorithm (i.e. increased mean and standard deviation in transmission latency as shown in Fig. 7) as our expense to favor short flows, which implies that the original algorithm is likely to over-penalize these very long flows.

We design P-SFD and S-SFD algorithms (presented in Section 3.3) to be less severe to long flows. We compare the performance of SFD, P-SFD, and S-SFD ($K = 4$) in Fig. 8(a). Both P-SFD and S-SFD algorithms can benefit more long

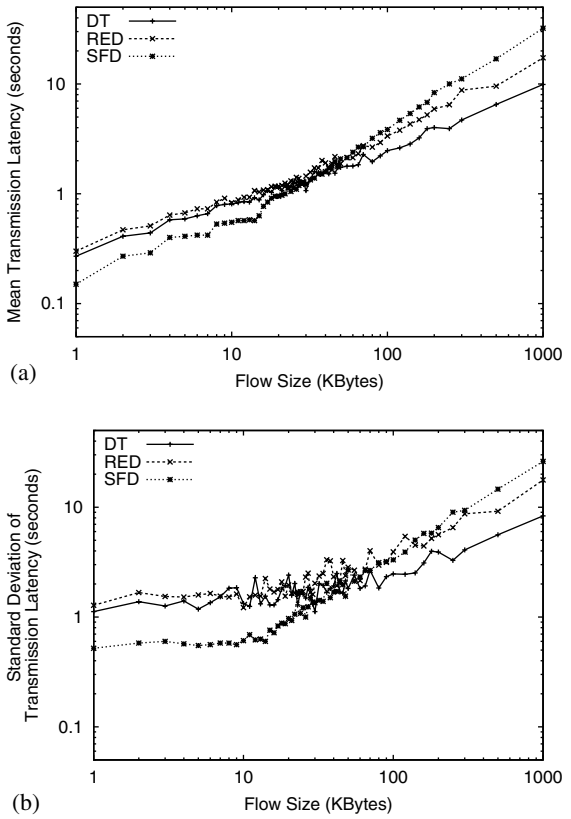


Fig. 7. Transmission latency of flows with size up to 1M bytes under different scenarios (in log–log scale): (a) mean transmission latency; (b) standard deviation of transmission latency.

flows; S-SFD algorithm shows the improved performance for flows up to about 300K bytes, corresponding to more than 98% of all web traffic flows. Compared to the basic SFD algorithm, S-SFD algorithm reduces the transmission latency for very long flows: the reduction is about 90% for 1M byte flow.

Since S-SFD shows greater improvement and is easier to implement, we believe S-SFD is a good choice to be deployed in networks. In the remaining part of this paper, we use S-SFD as the default SFD algorithm.

We also study the performance of SFD with different flow identification thresholds. The simulation results are shown in Fig. 8(b). We do not observe much difference in performance improvement with threshold set to 25 and 35K bytes,

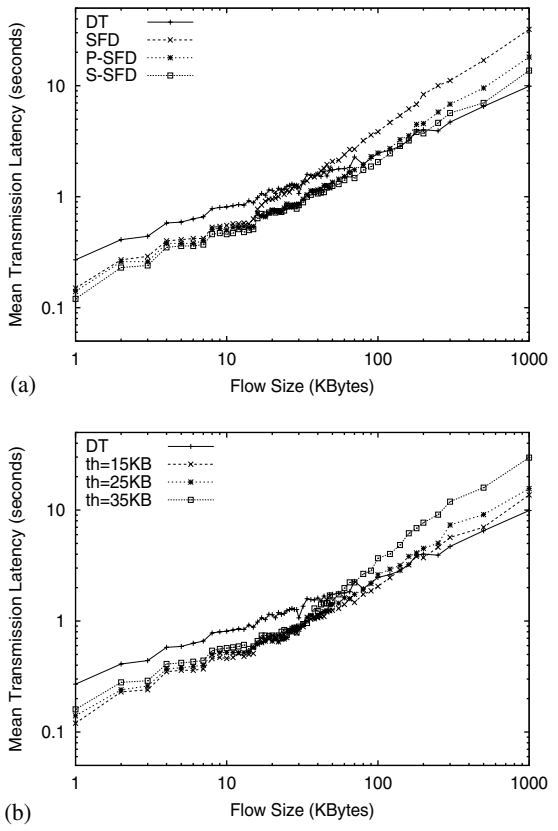


Fig. 8. Transmission latency with different algorithm configurations to reduce the penalty to long flows (in log–log scale): (a) P-SFD and S-SFD algorithms; (b) S-SFD algorithm with different identification thresholds.

which suggests that *15K bytes* is a reasonable threshold with the traffic pattern in our simulations. Ideally, the threshold should be able to adapt to traffic pattern observed.

4.4. Overall effects on end-user browsing

Although we have shown that SFD algorithm can reduce the transmission latency for more than 98% web objects (both from simulation and the estimation based on real network traces UCB96 and BU98), we can not conclude that the latency to retrieve web pages is also reduced because web pages may have different combinations of objects. For example, SFD is likely to penalize web pages with rich contents such as movies. To evaluate the overall effect that the SFD algorithm on end-user browsing, we need to quantify the changes in web page retrieval latency. We show some estimated results in this section.

Different implementations of browsers and web servers restrict the number of web objects which can be retrieved concurrently (i.e. the number of concurrent connections). We examine two extreme cases below:

- *Parallel retrieval*. The number of allowed concurrent connections is infinite: all web objects on the target page are transferred in parallel. In this case, the web page retrieval latency is always determined by the largest object embedded [34] and can be simply estimated as its transmission latency.
- *Sequential retrieval*. The number of allowed concurrent connection is 1: the web objects are transferred sequentially (similar to HTTP/1.1 with persistent connection). In this case, the web page retrieval latency can be estimated as the sum of transmission latency of all web objects embedded.

We believe that these two case studies give us some hints about the real web latency. They may also show bounds under certain circumstances.

We start with five representative types of web pages [34] listed in Table 5. We estimate page retrieval latency with S-SFD under both parallel and sequential retrieval schemes and compare the result with DT case. The absolute values and improvement (in percentage) are summarized in Table 6 (positive value means reduction in retrieval latency). We find that SFD algorithm can reduce the page retrieval latency for all these five types of web pages: by 15–77% in parallel retrieval case and by 31–43% in sequential retrieval case. We also observe that the improvement is not significant if the web page has a large object embedded (for example, the Frames page contains a 83K byte HTML part). This is because SFD increases transmission latency for large objects.

Since these five types of web page are not equally likely to be retrieved in reality, we further estimate web page retrieval latency from real trace UCB96. We compute page size (including all

Table 5
Five representative types of web pages

Page types	Web objects name	Web objects size (K bytes)	Number of web objects embedded
Text page	Large HTML part	29	1
	Medium images	7–13	3
Map page	Small HTML part	5	1
	Small images	1–3	3
	Large image	67	1
Graphics page	Medium HTML part	7	1
	Small images	1–3	9
	Medium images	9–12	4
Frames page	Small HTML parts	1–2	4
	Medium HTML part	7	1
	Large HTML part	83	1
	Small images	1–3	14
	Medium images	5–19	7
Java pages	Medium HTML part	8	1
	Small images	1–2	7
	Medium images	4–11	3
	Small Java parts	1–3	14
	Medium Java parts	4–18	11

objects on one page) by grouping requests with the same source and destination pair together. A request for a *html or htm* object is treated as the beginning of a new web page.

We calculate the changes (in percentage) with SFD algorithm under both parallel and sequential retrievals and plot the cumulative distribution function (CDF) in Fig. 9 (positive percentage means reduction in latency). We find that by applying SFD more than 90% web pages are transfered at least 30% faster while only less than 1% web pages show worse performance. Therefore, we

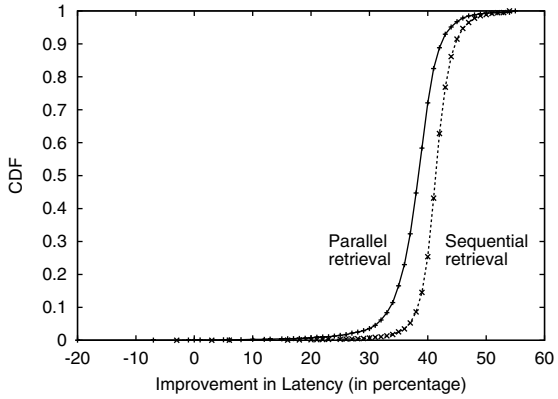


Fig. 9. CDF: improvement of web page retrieval latency with S-SFD algorithm.

conclude that the SFD algorithm reduces web latency in general.

We also notice that with HTTP/1.1 browsers and servers maintain persistent connections for request/response exchanges, which reduces the number of short flows since several web objects can be transferred by one connection. Recent measurement [7] shows a notable percentage (40–50%) of web objects are now transferred by persistent connections; but pipelining of request/response has not been supported by popular browsers yet. With HTTP/1.1, web objects are transferred by one long flow with short idle time intervals (1 s, for example) in between. Since SFD resets flow states after similar timeout periods (as discussed in Section 3.2), the transfer of different objects within a persistent connection will be treated as separate short flows. Therefore, we believe that the SFD algorithms should show similar improvement for HTTP/1.1 web traffic, although future work is needed to verify this claim.

Table 6
Retrieval latency (s) of different web pages

Page types	Parallel retrieval			Sequential retrieval		
	DT	S-SFD	Improvement (%)	DT	S-SFD	Improvement (%)
Text page	1.27	0.78	39	3.7	2.16	42
Map page	2.28	1.47	36	4.1	2.52	39
Graphics page	0.83	0.47	77	7.63	4.32	43
Frames page	2.21	1.87	15	15.92	9.47	41
Java page	1.03	0.69	33	20.88	14.43	31

4.5. Performance under different network loads

To examine the performance of SFD under other network loads, we repeat the above simulations in light and heavy network load scenarios, which are described in Table 3.

We show transmission latency of different flows in both scenarios in Fig. 10. We find that under light network load, neither RED or SFD gives considerable performance improvement; SFD performs very similarly as RED except very long flows are penalized. On the other hand, SFD shows dramatic performance improvement under heavy load: short flows are transmitted more than 10 times faster. Recall the comparison under medium network load shown in Fig. 7, we claim that SFD achieves larger performance improvement as network load increases. Intuitively, this observation is consistent with the idea of "network hole plugging": at light load there is little traffic so flow differentiation is irrelevant; but at heavy load it is easier to find holes between short flows that long flows can fill in.

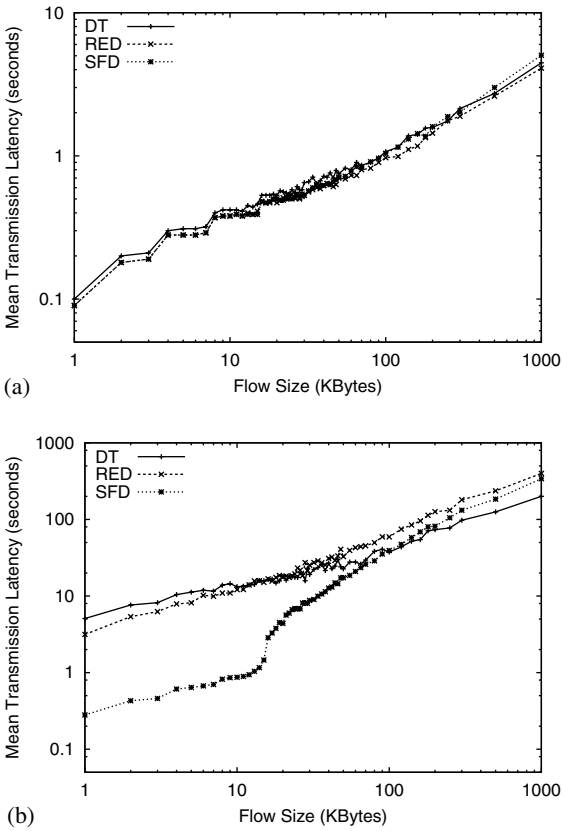


Fig. 10. Mean transmission latency under different network load (in log–log scale): (a) light loaded network; (b) heavy loaded network.

4.6. Sensitivity in simulation configurations

The above simulation results are from a very simple scenario: simple dumbbell topology with only web traffic. To investigate the sensitivity of our previous simulation, we relax the following aspects of the simulation scenario: network topology, various RTTs, presence of non-web traffic, and the effect of web server CPU model. While we can not claim to simulate "the Internet" [35], these results can help us to better understand dynamics of SFD algorithm.

Network topology. Choice of simulation topology is difficult. Rather than select an arbitrary topology from a real network, we focus on controlled studies of restricted topologies to understand specific network effects (as has been previously observed by Feldmann et. al. [17]). A limitation of a dumbbell topology is that there is no intermediate queuing in the network. To relax this limitation, we repeat simulations in a topology (shown in Fig. 11) with two tiers of clients being connected to router $R2$ with various link bandwidth and propagation delay. Thus, the second tier links could also become bottlenecks besides the link between $R0$ and $R1$.

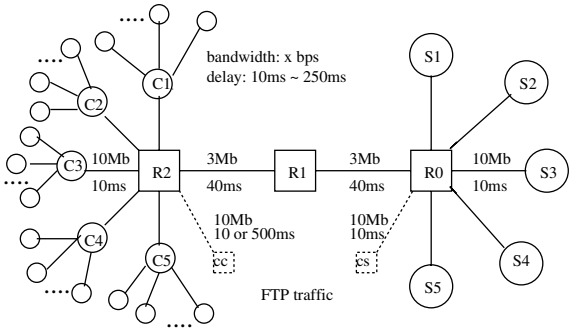


Fig. 11. A network with two tiers of clients.

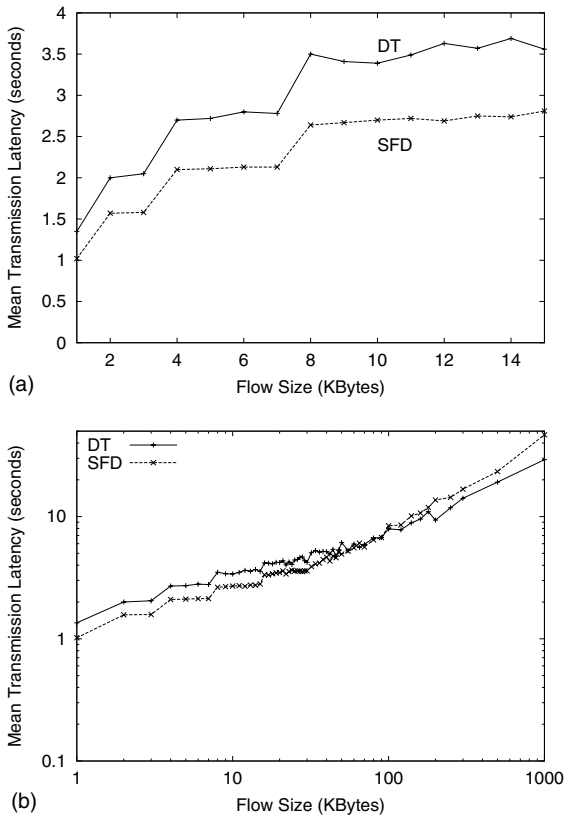


Fig. 12. SFD shows similar performance in a two-tiered topology with multiple bottleneck links: (a) SFD reduces transmission latency for short flows; (b) impact on long flows.

We show transmission latency of short and long flows for this configuration in Fig. 12. Similar to our previous results, SFD algorithm protects short flows and penalizes long flows. As shown in Table 4 (scenarios 2-tier), SFD reduces transmission latency for short flows by 23%. We also observe that transmission latency of very long flows (1000K bytes) increases by about 20%. We therefore conclude that intermediate queuing does not substantially change the performance of SFD algorithm.

Various RTTs. For short flows with same flow size, the performance improvement of SFD may vary as flows have different RTTs. To investigate this effect, we examine traffic in above simulations and group flows with same RTTs together. We compute the performance improvement of SFD (compared to DT) for each group. We find that the coefficient of correlation between RTTs and the corresponding improvements is about 0.3, which suggests that SFD performance improvements are weakly correlated with RTTs, with slightly more benefit for far flows (with large RTTs) than for near flows.

We have shown that SFD improves the performance for short flows. We also note that far flows have worse performance than near ones. Therefore, it is not surprising that short, far flows would have a larger *relative* improvement than others, since they are favored (short) and can be largely improved (far).

Presence of non-web traffic. To evaluate the performance of the SFD algorithm with presence of non-web traffic, we inject FTP traffic from node *cs* to node *cc* (refer to Fig. 11). The simulation results confirm that SFD algorithm protects short web flows effectively. It also penalizes FTP traffic similarly as long web flows: reducing their goodput by about 40%.

In the above study, we note that long flows with low rate (for example, long FTP flows with large RTTs) have little effect on short flows' transmission. However, SFD still penalizes these long flows because it differentiates flows only by their sizes. A potential variation could be to differentiate flows by their rates.

Server CPU delay. For all simulations above, we emphasize network queuing effect on the transmission latency of response flows. While network latency is our major concern in this paper, CPU delay is also important for busy web servers. We briefly investigate this effect below.

We add a simple CPU model to web servers in our simulation. We assume that the server CPU has infinite buffer size and constant processing rate that does not change under different load. We implement two simple scheduling policies: first-come, first-serve (FCFS) and shortest task first (STF). For STF policy, we assume that the server always knows the size of web object requested.

We simulate two scenarios: network-limited, where the bottleneck is in the networks, and server-limited, where servers process requests slowly. In each scenarios, we apply each of the four combinations of server and network scheduling policies: DT + FCFS, DT + STF, SFD + FCFS,

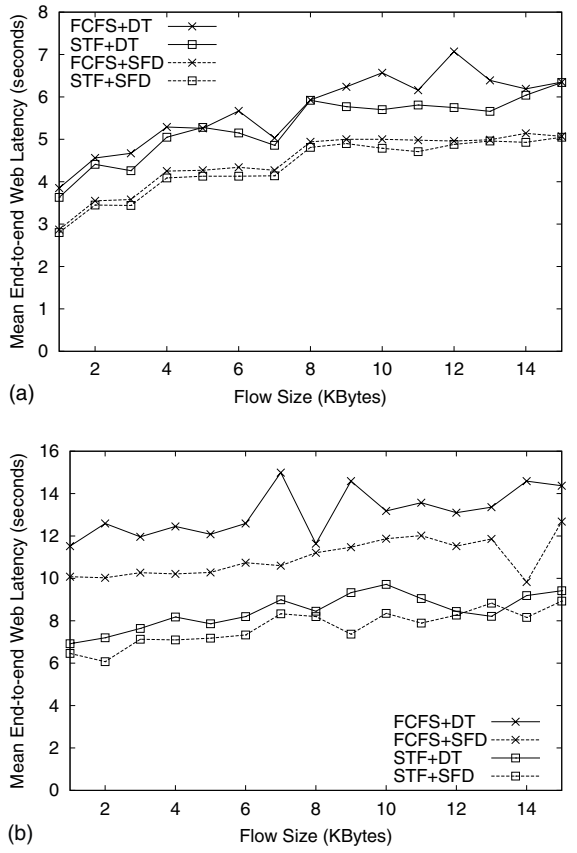


Fig. 13. The end-to-end latency of short web flows under different scenarios: (a) network-limited scenario; (b) server-limited scenario.

and SFD + STF. We measure the end-to-end web latency, that is, the time from client sending out a request till it receiving the last packet of corresponding response.

We show simulation results in Fig. 13 and summarize the improvement in Table 7. We find that the combination of SFD and STF always gives the best performance (lowest end-to-end web latency for short flows) in both network-limited and server-limited scenarios; while DT and FCFS always shows the worst. Further, the improvement by server scheduling policy is fairly small (less than 6%) in a network-limited scenario (Fig. 13(a)). Similarly, network scheduling policy has limited effect on the end-to-end web latency if the scenario is server-limited (Fig. 13(b)).

Table 7
Reduction in end-to-end web latency under different scenarios and scheduling policies

Scenarios	DT + STF (%)	SFD + FCFS (%)	SFD + STF (%)
Network-limited	6	21	23
Server-limited	35	16	41

5. Conclusion

In this paper, we investigate the interaction among short and long web traffic flows and show how this interaction affects transmission latency of short flows. We propose SFD algorithm to give preferential treatment to short flows so that its transmission latency and the overall web latency are reduced.

We evaluate our algorithms in simulations. The results are summarized below:

1. SFD algorithm reduces transmission latency for short flows by about 34% and shows much smaller variance. Since about 90% of web traffic flows have better performance, most web transactions are accelerated. SFD algorithm also reduces transmission latency for some flows with medium size.
2. We further evaluate SFD algorithm by estimating the web page retrieval latency. The result shows that more than 90% of web pages can be transfered faster with SFD algorithm.
3. Although some long flows can also benefit from SFD algorithm, the transmission latency of very long flows is increased. However, this penalty is well bounded by SFD. We further propose S-SFD and P-SFD algorithms to reduce the penalty to long flows. We prefer to use S-SFD algorithm because of its better performance and simplicity. We also study the effect of different identification thresholds in SFD algorithm.

We note that the performance improvement achieved by SFD is based on the assumption of the burntness of traffic: there are "holes" between the bursts of short flows where long flows can fill in. However, traffic in heavily loaded backbone links

shows Poisson arrivals [36]: there is no "hole" at all. In that case, SFD algorithm cannot improve the performance for backbone traffic.

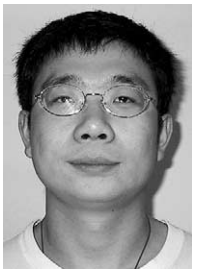
Acknowledgements

We would like to acknowledge the Advanced IP Networks group in Nortel Networks for their contribution of DiffServ model to *ns-2*, and Dr. Mark Crovella for pointing us to their traces (BU98). We appreciate the fruitful discussion with Deborah Estrin, Ted Faber, and Sally Floyd. We are also graceful to anonymous reviewers who have given us insightful comments on this work. This material is based upon work supported by DARPA via the Space and Naval Warfare Systems Center San Diego under Contract No.N66001-00-C-8066 ("SAMAN"), and by the CONSER project supported byNSF.

References

[1] J.B. Postel, Internet Protocol, RFC 791, Internet Request For Comments, 1981.
[2] W. Prue, J. Postel, A queuing algorithm to provide Type-of-Service for IP links, RFC 1046, Internet Request For Comments, February 1988.
[3] K. Nichols, V. Jacobson, L. Zhang, A two-bit differentiated services architecture for the Internet, draft-nichols-diff-svc-arch-00.txt, Internet Draft, December, 1997.
[4] S. Blake, D. Black, M. Carlson, E. Davies, W. Weiss, Z. Wang, An architecture for differentiated service, RFC 2475, Internet Request For Comments, December 1998.
[5] D. Clark, W. Fang, Explicit allocation of best effort packet delivery service, ACM/IEEE Transactions on Networking 6 (4) (1998) 362–373.
[6] N. Bhatti, A. Bouch, A. Kuchinsky, Integrating user-perceived quality into web server design, in: Proceedings of the International World Wide Web Conference, Amsterdam, May 2000, pp. 1–16.
[7] F.D. Smith, F. Hernandez Campos, K. Jeffay, D. Ott, What TCP/IP protocol headers can tell us about the web, in: Proceedings of the ACM SIGMETRICS, Cambridge, MA, June 2001, pp. 245–256.
[8] X. Zhu, J. Yu, J. Doyle, Heavy tails, generalized coding, and optimal web layout, in: Proceedings of the IEEE Infocom, Anchorage, Alaska, USA, April 2001, IEEE.
[9] T. Berners-Lee, R. Fielding, H. Frystyk, Hypertext transfer protocol—HTTP/1.0, RFC 1945, Internet Request For Comments, May 1996. Available from <ftp://ftp.isi.edu/in-notes/rfc1945.txt>.
[10] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext transfer protocol—HTTP/1.1, RFC 2616, Internet Request For Comments, June 1999. Available from <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.
[11] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the Internet, ACM/IEEE Transactions on Networking 7 (4) (1999) 458–473.
[12] Cooperative Association for Internet Data Analysis, Netramet web session performance project. Available from <http://www.caida.org/analysis/workload/netramet/web/>.
[13] M.E. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, in: Proceedings of the ACM SIGMETRICS, Philadelphia, Pennsylvania, May 1996, ACM, pp. 160–169.
[14] P. Barford, M. Crovella, Generating representative web workloads for network and server performance evaluation, in: Proceedings of the ACM SIGMETRICS, Madison, WI, USA, June 1998, ACM, pp. 151–160.
[15] P. Barford, M.E. Crovella, Measuring web performance in the wide area, ACM Performance Evaluation Review 27 (2) (1999) 37–48.
[16] P. Barford, A. Bestavros, A. Bradley, M.E. Crovella, Changes in web client access patterns: characteristics and caching implications, World-Wide Web Journal 2 (1999) 15–28.
[17] A. Feldmann, A.C. Gilbert, P. Huang, W. Willinger, Dynamics of IP traffic: a study of the role of variability and the impact of control, in: Proceedings of the ACM SIGCOMM, Cambridge, MA, USA, August, 1999, ACM, pp. 301–313.
[18] V.N. Padmanabhan, J. Mogul, Improving HTTP latency, in: Proceedings of the International World Wide Web Conference, Chicago, IL, USA, October 1994, pp. 995–1005.
[19] H.F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H.W. Lie, C. Lilley, Network performance effects of HTTP/1.1, CSS1, and PNG, in: Proceedings of the ACM SIGCOMM, Cannes, France, September 1997, ACM, pp. 155–166.
[20] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, R.H. Katz, TCP behavior of a busy Internet server: analysis and improvements, in: Proceedings of the IEEE Infocom, San Francisco, CA, USA, March 1998, IEEE, pp. 252–262.
[21] L. Eggert, J. Heidemann, Application-level differentiated services for web servers, World-Wide Web Journal 2 (3) (1999) 133–142.
[22] M.E. Crovella, R. Frangioso, M. Harchol-Balter, Connection scheduling in web servers, in: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Boulder, CO, USA, October 1999.
[23] N. Bansal, M. Harchol-Balter, Analysis of SRPT scheduling: investigating unfairness, in: Proceedings of the ACM SIGMETRICS, Cambridge, MA, USA, June 2001, ACM, pp. 279–290.

[24] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair-queueing algorithm, in: Proceedings of the ACM SIGCOMM, Austin, TX, USA, September 1989, ACM, pp. 1–12.
[25] P. Hurley, M. Kara, L. Boudec, P. Thiran, ABE: providing a low-delay service within best-effort, IEEE Network Magazine 15 (3) (2001) 60–69.
[26] R. Mahajan, S. Floyd, Controlling high bandwidth flows at the congested router, in: Proceedings of the IEEE International Conference on Network Protocols, Riverside, CA, USA, November 2001, IEEE, pp. 1–12.
[27] L. Guo, I. Matta, The war between mice and elephants, in: Proceedings of the IEEE International Conference on Network Protocols, Riverside, CA, USA, November 2001, IEEE.
[28] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, ACM/IEEE Transactions on Networking 1 (4) (1993) 397–413.
[29] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, Technical Report, ICSI, 2001.
[30] M. Allman, S. Floyd, C. Partridge, Increasing TCP's initial window, RFC 2414, Internet Request for Comments, September 1998, Available from: <http://www.w3.org/Protocols/rfc2414/rfc2414.txt>.
[31] VINT group, UCB/LBNL/VINT network simulator—ns (version 2). Available from <http://www.isi.edu/nsnam/ns/>.
[32] M. Christiansen, K. Jeffay, D. Ott, F.D. Smith, Tuning RED for web traffic, in: Proceedings of the ACM SIGCOMM, Stockholm, Sweden, August 2000, pp. 139–150.
[33] Lawrence-Berkeley Labs, The Internet traffic archive, Available from: <http://ita.ee.lbl.gov/>.
[34] L. Eggert, J. Heidemann, J. Touch, Effects of ensemble-TCP, ACM Computer Communication Review 30 (1) (2000) 15–29.
[35] S. Floyd, V. Paxson, Difficulties in simulating the Internet, ACM/IEEE Transactions on Networking 9 (4) (2001) 392–403.
[36] J. Cao, W.S. Cleveland, D. Lin, D.X. Sun, On the nonstationarity of Internet traffic, in: Proceedings of the ACM SIGMETRICS, Cambridge, MA, June 2001, pp. 102–112.



Xuan Chen is a graduate research assistant at USC/ISI and a Ph.D. candidate at Computer Science Department, USC. His research interests include adaptive queue management and robust network traffic control under heavy load. He received his B.E. from Beijing University of Posts and Telecommunications and his M.S. from USC. He is a student member of ACM.



John Heidemann is a project leader at USC/ISI and a research assistant professor at USC. At ISI he investigates networking protocols and simulation as part of the SAMAN and CONSER projects and embedded networking and sensor networking as part of the SCADDS project. He received his B.S. from University of Nebraska-Lincoln and his M.S. and Ph.D. from UCLA, and is a member of ACM, IEEE, and Usenix.