SIFT: A low-complexity scheduler for reducing flow delays in the Internet

CENG-2004-01, University of Southern California

Konstantinos Psounis
USC
kpsounis@usc.edu

Arpita Ghosh
Stanford University
arpitag@stanford.edu

Balaji Prabhakar
Stanford University
balaji@stanford.edu

ABSTRACT

Bandwidth is allocated to a flow in today's Internet due to actions by transport protocols at end-systems and queue management schemes at routers. While the nature of this bandwidth allocation is not yet well understood, to minimize the average flow delay, it would be ideal to allocate bandwidth according to a policy like SRPT (shortest remaining processing time). Given the heavy-tailed nature of Internet flow size distribution, the corresponding reduction in average flow delay would be particularly pronounced. But, this ideal is impractical: to decide which packet to transmit next a router would now need to know the residual data in the flow corresponding to each currently buffered packet. Even a less complex cousin of SRPT, like SFF (Shortest Flow First), is unimplementable since it would require a knowledge of flow sizes at routers.

In this paper, we introduce a randomized algorithm called SIFT, for separating the packets of long and short flows. It is based on the simple observation that a randomly sampled arriving packet is much more likely to belong to a large flow, allowing a router to differentially allocate link bandwidth to large and small flows. We analyze the performance of SIFT using queueing models, comparing its performance with FIFO and PS (processor sharing). We also compare its performance with packet-level FIFO, the current practice in Internet routers, via ns simulations. We find that SIFT reduces the delay of the vast majority of flows by one to two orders of magnitude without significantly increasing the delay of the longest flows. We comment on the implementability of SIFT and argue that it is feasible to deploy it in today's Internet.

1. INTRODUCTION

Scheduling policies significantly affect the performance of resource allocation systems. The policy that services the job with the shortest remaining processing time (SRPT) is known to minimize average response times [19]; the improvement over the first-in-first-out (FIFO) discipline is tremendous, especially when job sizes are drawn from a heavy-tailed distribution. SRPT is also significantly better than the processor sharing (PS) discipline. Indeed, SRPT has been used by researchers to exploit the heavy-tailed nature of web traffic [7] and the CPU requirements of processes [12, 15], to yield orders of magnitude improvement in the mean delay of real systems compared with the FIFO and PS discplines [11].

Given the heavy-tailed nature of Internet flow-size distribution [21], one hopes that scheduling the packets of flows using SRPT at routers would lead to a significant reduction in the delay of Internet flows. But, it is not obvious what it means to employ SRPT in the Internet. For example, in contrast to the web server and CPU environments, flows don't just share a single resource: they share bandwidth at multiple links, and with different sets of flows at each link. One sensible interpretation of SRPT in the Internet might be for each router to next transmit a packet from that flow which has the least residual amount of data. This scheme is clearly impractical: It would require routers to know how much data is left in the flow corresponding to each currently buffered packet.

Recognizing this difficulty, researchers have considered servicing a packet from the flow that has the least attained service (LAS) [23],[18]. This policy has also been considered in detail in the classical queueing literature, where it has been referred to as the Foreground-Background (FB) scheduler [13], [20]. While not performing as well as SRPT, it has been shown that LAS and FB work well in reducing average delay when job or flow sizes are heavy-tailed. But, the FB and LAS policies require routers to maintain per-flow state, and potentially maintain per-flow queues. The sheer number of flows on a high speed link is so large [6, 9] as to make it impractical for routers to keep per-flow state.

A packet service policy like shortest flow first (SFF) is simpler than SRPT or LAS because the classification of flows is a one-time affair, there is no need to dynamically update the residual or attained service. However, routers don't know flow sizes! And, even if they did, per-flow queueing may be required to serve the next packet by consulting a list of flows sorted by size.

In this paper we propose a novel randomized scheme called SIFT that separates the packets of long and short flows (this explains its name), and preferentially serves the latter. Flows are distinguished by randomly sampling arriving packets. Since large flows have many packets, SIFT will

.

identify them with a very high probability. Once identified, the packets of long flows are enqueued in a low priority buffer, while the packets of short flows are preferentially served from a high priority buffer. If a packet has been put into the low priority buffer, to avoid mis-sequencing, further packets from its flow will also be put into the low priority queue.[1]

Having thus described our scheme, we note that its implementation requirements are pretty low (implementation is discussed in detail in Section 5). Indeed, the scheme requires two queues, a priority mechanism for servicing them, and a sampling mechanism at the packet level. It assumes that the router where it is deployed can identify which flow each packet belongs to.[2]

The major problem is to understand how well SIFT performs in reducing the average delay. We compare performance in two ways: via queueing analysis and via ns simulations. Analyzing a single queue at which heavy-tailed jobs arrive, we find that SIFT performs orders of magnitude better than FIFO, significantly better than PS, and that it is closer to SRPT than it is to FIFO or PS.[3] Simulations performed with ns tell us that, in the context of Internet flows and depending on the load, SIFT reduces the average delay of short flows between one and two orders of magnitude, and the overall average delay between two and ten times.

A common concern with SRPT, or other pre-emptive disciplines, is the danger of starving long jobs. The authors in [3] show that this is rarely the case for SRPT under realistic heavy-tailed distributions. Our simulations also show that SIFT increases the delays of long flows by at most a factor of 2 or 3. To completely eliminate the possibility of starvation, we modify SIFT so that the low priority queue is guaranteed a small percentage of the link capacity. Interestingly, simulations show that this modification doesn't degrade the performance of SIFT.

The organization of the paper is as follows: In Section 2 we describe SIFT in detail. Section 3 provides an analysis of the proposed scheme using queueing theory. In Section 4 we present the results of simulations with TCP flows using ns-2 [17]. Section 5 comments on the implementation requirements of SIFT and deployment issues, and Section 6 concludes the paper.

2. DESIGNING A LOW-DELAY SCHEDULER FOR INTERNET FLOWS

In this section we discuss in detail the difficulties with implementing schemes like SRPT, LAS, or SFF in today's Internet. This discussion motivates the design of SIFT, which we describe at the end of the section.

We start with the obstacles faced when attempting to implement SRPT. First, it is not clear what it means to deploy SRPT in the Internet. In the following discussion we envision a router that implements SRPT by keeping track of the number of the remaining packets to be serviced for each flow that goes through it.

To determine this number, the router needs to know the size of the flow at the time of arrival of its first packet, and to maintain throughout the duration of the flow's life the number of its packets that have already been serviced. However, as mentioned in the introduction, flow sizes are not known at the time of arrival, and the logistics associated with counting packets for every flow are enormous. Even if the remaining processing time of all flows could be determined, it is possible that the flow with the shortest remaining processing time does not have any packets queued at the router at some point in time, and hence its remaining packets are not available for service.

Suppose even that the number of remaining packets of a flow was known, and that the scheduler would service the packets of the flow with the shortest remaining processing time among the flows that currently have packets in the router (rather than among all active flows). Then, one would implement SRPT by maintaining a separate queue for each active flow, and employing a strict priority rule between the queues. The highest priority queue would be the one that corresponds to the flow with the shortest remaining processing time, the second highest priority queue would correspond to the flow with the second shortest remaining processing time, and so on. If at some point in time the highest priority queue were empty, the router would service packets from the second highest, etc. But this scenario also has a problem: the number of concurrently active flows on the Internet is so large that per-flow queueing is impractical. For all of these reasons, it is not possible to implement exact SRPT in the context of Internet flows.

The LAS scheme does not require the knowledge of flow sizes. However, it does require to maintain per-flow state, in particular, to keep track of the number of packets that have been serviced so far from each flow. SFF does not require to maintain per-flow state, but it needs the knowledge of flow sizes. Finally, both LAS and SFF would require per-flow queueing in order to be implemented, similar to SRPT. Hence, it is not feasible to use either of these schemes in the Internet.

We will now address the problems mentioned above: per-flow queueing requirement, non-availability of packets at service times, per-flow state maintenance, and unknown flow sizes at arrival times.

A simple solution to the per-flow queueing problem is to reduce the number of queues to two; one high priority queue for the "short" flows and one low priority queue for the "long" flows. Having only two queues will not hurt the performance much for the following reason: since the flow size distribution is heavy-tailed, the troublesome flows are the few very long ones. Hence, it suffices to place *these* flows on a low priority queue to significantly reduce delays.

Notice that by using two queues in this fashion, one can sidestep the non-availability of packets problem as follows: The scheme now prioritizes queues rather than flows. For serving each individual queue, one can choose a work conserving service discipline that serves currently enqueued packets, rather than a discipline which might wait to serve packets of a particular flow.

[1] This is reminiscent of the "sample and hold" strategy advanced in [5] for identifying and counting the packets of high bandwidth flows.

[2] Notice that in accordance with the usual practice [6, 8, 9], packets are said to belong to the same flow if they have the same source and destination IP address, and source and destination port number. A flow is "on" if its packets arrive more frequently than a certain "timeout" number of seconds. The timeout is usually set to something less than 60 seconds.

[3] It is to be remembered that while implementability improves in moving from SRPT to SFF to SIFT, the reduction in average delay drops. Thus, the performance of SIFT cannot be expected to be comparable to that of SRPT.

Schemes maintain per-flow state and require the knowledge of flow sizes in order to decide which flow to service first. To simplify matters, we follow the spirit of SFF and consider only flow sizes to order flows. To decide which flow is short and which is long, we sample every arriving packet independently with the same probability and store the flow id of the sampled flows together with the total number of sampled packets of each sampled flow. Once this number exceeds some threshold, we forward all future packets of the flow to the low priority queue. We call such a flow long. Short flows are the flows all of whose packets stay in the high priority queue. From an implementation point of view, choosing a sample threshold equal to one is the best. In Section 3.2 we investigate how different sample thresholds affect the performance of the scheme. It turns out that a threshold of one has quite good performance, hence, this is what we use.

As mentioned in the introduction, we shall refer to the proposed scheme as SIFT. Figure 1 shows the scheme in pseudo-code and Figure 2 presents it schematically. Note that if the original queue has size B, the size of the high plus the low priority queue should not exceed B.

```
if (packet_arrival) {
   flow_id = get_flowid_from_packet;
   if (sampled_packets(flow_id) > threshold) {
      forward_to_low_priority_queue;
   } else {
      forward_to_high_priority_queue;
      if (sample_flow == true)
         sampled_packets(flow_id)++;
   }
}
```

Figure 1: SIFT in pseudo-code.

Remarks: When the flow data arrive in a packetized fashion spread in time, scheduling decisions based only on flow sizes, or on the number of the remaining packets requiring service, are suboptimal. To see this, suppose an oracle knows the arrival times of all future packets of all flows. Then, to minimize delays, the oracle would assign the highest priority to the packets of the flow that would finish service nearest in the future than any other flow under this priority. Which flow this is depends on both the number and the arrival times of its remaining packets. If, for example, some of the packets of the shortest flow were to arrive late, then servicing the packets of the second shortest flow first might yield lower average delay.

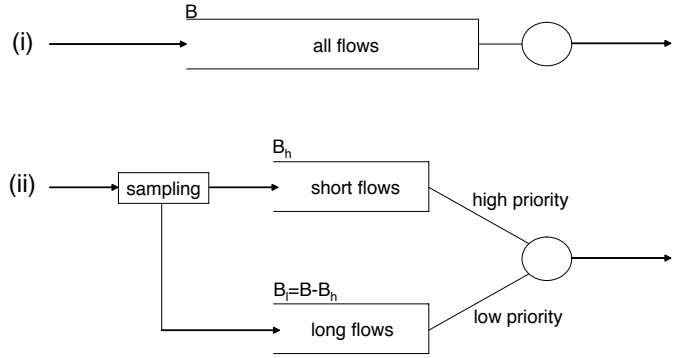

Figure 2: (i) original queue, (ii) SIFT.

Even though SIFT is a flow scheduling mechanism, it has implications to congestion control. To see this, consider the deterministic version of SIFT, that is, a scheme that uses a size threshold T, and forwards the packets of the flows with size less than T to the high priority queue, and the rest to the low priority queue. Then, depending on the value of T, this scheme treats differentially flows that stay in slow start phase throughout their existence, from flows that enter congestion avoidance. Further, since SIFT starts forwarding packets of a flow to the low priority queue only after one packet has been sampled, it can also be thought of as randomizing the deterministic scheme that forwards the first $T-1$ packets of *all* flows to the high priority queue and the rest to the low priority queue. Such a scheme treats preferentially all packets that are sent during slow start. It is outside the scope of this paper to further investigate the connections between SIFT and congestion control. We leave this as future work.

3. ANALYZING SIFT

In this section, we analyze the performance of SIFT using queueing theory. The goal is to investigate the gains from sifting short flows from the rest of the traffic[4] under the simplifying assumption that all the packets of a flow arrive at the same time. We use a single link setup to compare the average response time of SIFT to that of a FIFO or PS queue. We also compare its performance to that of SRPT in order to understand how far it is from the optimum.

Even though this analysis cannot substitute for simulations that capture in detail TCP and network dynamics, it is not irrelevant to practice since the flow scheduling discipline for flows in slow start (congestion avoidance) can be roughly approximated by FIFO (PS) [10].

3.1 The Model

Flows arrive to the link as a Poisson process of rate λ. The size of a flow, measured in number of packets, follows a bounded Pareto distribution specified by three parameters: m, the smallest flow size, M, the largest flow size, and α, the shape parameter. The probability that an arriving flow has size (equivalently, service requirement) x is given by

$$P_p(x) = c_p x^{-\alpha-1} \qquad m \le x \le M, \tag{1}$$

where c_p is a normalization constant, chosen so that $\sum_m^M P_p(x) = 1$. As with the unbounded Pareto distribution, the value of α determines how heavy-tailed the distribution is: the closer α is to 1, the more heavy-tailed the distribution. Since we use a bounded distribution, both the mean and the variance of the flow sizes are finite. (Note that the unbounded Pareto distribution has finite mean but infinite variance for α between 1 and 2.)

We will now obtain the arrival rate and the size distribution of flows for the high and the low priority queues.

Without loss of generality, assume the sampling threshold equals one. A flow is said to be sampled if at least one of its packets are sampled. Let p be the packet sampling

[4]We will use "flows" instead of the more common term "jobs".

probability. Then, the probability that a flow of size x is sampled equals

$$P_s(x) = 1 - (1-p)^x. \quad (2)$$

Now, assume that all the packets of a sampled flow join the low priority queue while the rest join the high priority queue. The flows of a given size, say x, form a Poisson arrival process with rate $\lambda P_p(x)$. Since a flow of size x is sampled with probability $P_s(x)$ independent of all else, the arrival rate of flows of size x into the low priority queue is $\lambda_{L,x} = \lambda P_p(x) P_s(x)$. The total arrival rate into the low priority queue is therefore

$$\lambda_l = \sum_m^M \lambda P_s(x) P_p(x),$$

and the arrival rate into the high priority queue is

$$\lambda_h = \lambda - \lambda_l = \sum_m^M \lambda (1 - P_s(x)) P_p(x).$$

The above analysis makes two unrealistic assumptions. First, it assumes that all the packets of a flow arrive at the same time, that is, TCP dynamics are ignored. Second, it assumes that all the packets of a sampled flow join the low priority queue. It is very hard to account for TCP dynamics, but getting rid of the second assumption is actually easy.

Recall that under SIFT, once a packet of a flow is sampled, then that packet and further packets from the flow are put into the low priority queue, while preceding packets are put into the high priority queue. Thus, the probability that a flow arriving to the low priority queue has size x is

$$P_l(x) = \sum_{y \geq x} P_p(y)(1-p)^{(y-x)} p. \quad (3)$$

This is because a flow of size y ($y \geq x$) has only x of its packets in the low priority queue if the $(y-x+1)$th packet is the first one to be sampled. Similarly, the probability that a flow arriving to the high priority queue has size x is

$$P_h(x) = P_p(x)(1-p)^x + \sum_{y > x} P_p(y)(1-p)^x p. \quad (4)$$

In the above analysis, it is as if we partition sampled flows into two parts. The first part corresponds to packets arriving before the sampled packet and constitutes a flow that joins the high priority queue, and the second part corresponds to the rest of the packets and constitutes a flow that joins the low priority queue.

By taking into account that some of the packets of the sampled flows join the high priority queue, we introduce dependencies in the arrival times of flows that join the high and the low priority queues. To keep the analysis tractable, we ignore this dependency and assume independent flow arrival times. In particular, we assume that low priority flows arrive as a Poisson process with rate λ_l as before, and high priority flows arrive as a Poisson process with rate $\lambda_h = \lambda$, with size distributions given by Equations (3) and (4) respectively.

3.2 Preliminary Analysis

In this section we study the effect that different values of the sampling threshold have on SIFT's performance. Then, we compute the probability that SIFT misclassifies short flows as long flows and visa versa. (Note that in the rest of the paper, when we refer to a short flow, we refer to a flow all of whose packets are not sampled, while in this section, a short flow is a flow with size less than a given size threshold.)

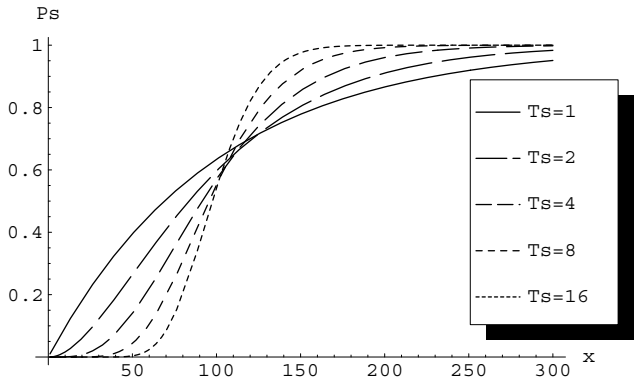

Figure 3: Flow sampling probability as a function of flow size.

Let T denote the size threshold used by a deterministic scheme to partition flows in short and long. Since SIFT classifies flows in a randomized fashion, it is reasonable to choose the packet sampling probability and the sampling threshold such that it takes T packets on average for a flow to be sampled, or equivalently, such that the expected number of packets till a flow is sampled equals T.

Let T_s denote the sampling threshold, and N denote the number of packets till a flow is sampled. For $T_s = 1$ it is easy to see that N follows a geometric distribution with average $1/p$. Hence, p should equal $1/T$. For $T_s > 1$, a flow is sampled if at least T_s of its packets are sampled. It is easy to see that in this case $E(N) = T_s/p$ and hence $p = T_s/T$. As T_s further increases this procedure converges to the deterministic scheme that "samples", i.e. forwards to the low priority queue, all flows whose size is at least T, since for $T_s = T$ the sampling probability equals one.

Figure 3 plots $P_s(x)$, the probability that a flow of size x is sampled, as a function of x for various values of T_s. T is set to 100. Note that $P_s(x) = 1 - \sum_{i=0}^{T_s-1} \frac{x!}{i!(x-i)!} p^i (1-p)^{x-i}$ which reduces to Equation (2) for $T_s = 1$. From the figure it is evident that the higher the value of T_s, the better the classification.

From an implementation point of view it is best to use $T_s = 1$. However, small values of T_s increase the probability that SIFT misclassifies flows. Let P_m denote this probability. P_m equals the probability that short flows are sampled plus the probability that long flows are not sampled, that is,

$$P_m = \sum_{x<T} P_s(x) P_p(x) + \sum_{x \geq T} (1 - P_s(x)) P_p(x).$$

For heavy-tailed flow size distributions P_m is quite small even for small values of T_s. For example, for $T_s = 1$, $p = T_s/T = 0.01$, $m = 1$, $M = 10^6$, and $\alpha = 1.1$, P_m equals 2.6%, that is, the fraction of misclassified flows is only 2.6%. The fraction of short misclassified flows among all short flows can be computed by $\sum_{x<T} P_s(x) P_p(x) / \sum_{x<T} P_p(x)$ and it equals 2.5%, and the fraction of long misclassified

flows among all long flows, computed in a similar manner, is 15.8%. The reason why there are so few flows that get misclassified, despite the non-negligible probability of sampling a flow of size less than T, is that the vast majority of flows are quite small, and the sampling probability for these flows is very small. Further, the sizeable fraction of misclassified long flows does not hurt performance much; as long as the very long flows are sampled performance is good, and this occurs with very high probability. For these reasons, in the rest of the paper we use a sample threshold equal to one.

3.3 Delay under SIFT with FIFO queues

In this section we compare the total, short-flow, and long-flow average delays under SIFT when it uses FIFO within each of its two queues, versus that under a single queue employing FIFO. We assume strict, preemptive priority between the high and low priority queues and do not place any limit on the buffer size of either queue.

Recall that $S \sim P_p(.)$, $S_h \sim P_h(.)$ and $S_l \sim P_l(.)$ are, respectively, the size distributions of a generic flow, a high-priority flow and a low-priority flow. Let D_h and D_l denote the delay of a high-priority flow and a low-priority flow, respectively.

When all flows are queued at a single FIFO queue, by the Pollaczek-Khintchine formula [22], the mean *waiting time* of each flow (short or long) equals $\frac{\lambda E(S^2)}{2(1-\rho)}$, where $\rho = \lambda E(S)$. The mean delay for each of type of flow is easy to compute and is stated in the proposition below.

PROPOSITION 1. *The short-flow and long-flow average delays in an M/G/1 FIFO queue are as follows:*

$$\begin{aligned} E_{FIFO}(D_h) &= E(S_h) + \lambda E(S^2)/2(1-\rho), \text{ and} \\ E_{FIFO}(D_l) &= E(S_l) + \lambda E(S^2)/2(1-\rho), \end{aligned}$$

where $\rho = \lambda E(S)$.

We now consider the mean delays under SIFT.

PROPOSITION 2. *The short flow (high priority) and long flow (low priority) average delays in an M/G/1 queue with a preemptive-resume discipline, and FIFO within each class are as follows:*

$$\begin{aligned} E_{SIFT,FIFO}(D_h) &= E(S_h) + \lambda_h E(S_h^2)/2(1-\rho_h), \text{ and} \\ E_{SIFT,FIFO}(D_l) &= \frac{\lambda E(S^2)}{2(1-\rho_h)(1-\rho)} + \frac{E(S_l)}{(1-\rho_h)}, \end{aligned}$$

where $\rho_h = \lambda_h E(S_h)$.

Proof: Follows directly from Chapter 10 in [22]. ■

Note that the average delay for the high priority flows under SIFT (Proposition 2) involves the second moment of the high priority flows, whereas in the single queue FIFO (Proposition 1) it involves the second moment of the *entire* distribution, which is very high for heavy-tailed distributions. Hence, SIFT makes a large reduction in the average delay of short flows. Also note that $E_{SIFT,FIFO}(D_l) = E_{FIFO}(D_l)/(1-\rho_h)$. Therefore, the average delay for a long flow under SIFT is worse by a factor $\frac{1}{(1-\rho_h)}$ than under a single FIFO.

A quantitative sense for the above formulas may be obtained from Figure 4. The values of the parameters are chosen as: $m = 1$, $M = 10^6$, $\alpha = 1.1$, and $p = 0.01$. The maximum value of the load, ρ, equals 0.9986.

With this parameter choice, 97.2% of the flows are short. As evident from the plot, SIFT with FIFO queues yields orders of magnitude lower delays for short flows without noticeably increasing the delays of long flows. Its overall average delay is significantly lower than that of FIFO.

3.4 Delay under SIFT with PS queues

Let us now consider the case where each of the two queues in SIFT uses PS as its service discipline. We will compare the average delays obtained in this case with the average delays under a single PS queue.

PROPOSITION 3. *The short flow and long flow average delays in a single M/G/1 PS queue are as follows:*

$$\begin{aligned} E_{PS}(D_h) &= E(S_h)/(1-\rho), \text{ and} \\ E_{PS}(D_l) &= E(S_l)/(1-\rho). \end{aligned}$$

Proof: It is well known that the expected delay of a flow of size x under PS equals $x/(1-\rho)$ (see, for example, [22]). The result follows by averaging over the appropriate sizes. ■

To the best of our knowledge, there is no known expression for the average delay of the low priority jobs in an $M/G/1$ queue with strict, preemptive priorities, and that uses PS within priority classes. In the following proposition we will derive a simple approximation to this expression using a very similar methodology to that used in Chapter 10 of [22] for the FIFO case.

We first state the following useful lemma whose proof can be found in Chapter 8 of [22]:

LEMMA 1. *The expected duration of an exceptional first service busy period (EFSBP),* B_e, *for an M/G/1 queue with an exceptional first service* S_e, *ordinary service* S *and traffic intensity* $\rho = \lambda E(S)$ *equals*

$$E(B_e) = E(S_e)/(1-\rho). \tag{5}$$

PROPOSITION 4. *The average delays of short (high priority) and long (low priority) flows in an M/G/1 queue with preemptive-resume priority discipline, and that uses PS within each priority class are:*

$$\begin{aligned} E_{SIFT,PS}(D_h) &= E(S_h)/(1-\rho_h), \text{ and} \\ E_{SIFT,PS}(D_l) &\approx \frac{E(S_l)}{(1-\rho)} + \frac{\lambda_h E(S_h^2)}{2(1-\rho_h)^2}. \end{aligned}$$

Proof: As far as the high priority flows are concerned, this system is the same as a standard $M/G/1$ queue with PS, where the only traffic that matters is the short flows. The average response time for these flows is therefore $E(S_h)/(1-\rho_h)$.

Let T_l be the duration of time from when a tagged low priority flow arrives to the system until it is serviced for the first time, and R_l be the duration of time from when it is serviced for the first time until it completes service. Thus, $D_l = T_l + R_l$.

T_l is the first time the system becomes clear of high priority flows after the tagged low priority flow arrived to the system. T_l is therefore an EFSBP. The exceptional first service is the time it takes to empty the high-priority work, V_h, seen by the tagged flow upon arrival. The relevant traffic intensity is ρ_h, since it is the new high priority flows that extend the busy period. Now, for any work conserving queue discipline, V_h is the same with that under FIFO. Hence, by Pollaczek-Khintchine, $E(V_h) = \lambda_h E(S_h^2)/2(1-\rho_h)$. Therefore,

$$E(T_l) = \frac{E(V_h)}{(1-\rho_h)} = \frac{\lambda_h E(S_h^2)}{2(1-\rho_h)^2}. \quad (6)$$

R_l can begin only when the system is clear of high priority flows, and ends when the tagged flow has finished service and the system is clear of high priority flows. So R_l is also an EFSBP with exceptional first service equal to the time the tagged flow spends being served, and relevant traffic intensity ρ_h as before.

The time spent by a low priority flow being serviced would have expected value $E(S_l)/(1-\rho_l)$, where $\rho_l = \lambda E(S_l)$, *if the server were working at rate 1.* [5] However, because of the preemptive priorities, the server works on the low priority flows at a rate $(1-\rho_h)$. So, the mean time spent by a low priority flow being serviced is $\frac{E(S_l)}{(1-(\rho_l/(1-\rho_h)))}$, and

$$E(R_l) \approx \frac{E(S_l)}{(1-(\rho_l/(1-\rho_h)))} \frac{1}{(1-\rho_h)} = \frac{E(S_l)}{(1-\rho)}. \quad (7)$$

The result is now obtained from combining Equations (6) and (7). ■

We use the formulae from Propositions 3 and 4 to plot the short-flow and long-flow average delays under a single PS queue, and under SIFT with PS queues. We also average over the two classes of flows to obtain the overall delay for both schemes. Figure 5 plots the results for $0 < \rho < 1$, $m = 1$, $M = 10^6$, and $\alpha = 1.1$. As before, 97.2% of the flows are short, and the rest are long.

SIFT is significantly better than PS for short flows under heavy load. This is expected since the average delay for the high priority flows under PS is larger by a factor $(1-\rho_h)/(1-\rho)$ than under SIFT, and this factor is very large for ρ close to 1. The performance of the two schemes for long flows is very similar. Finally, SIFT performs better than PS with respect to overall delay at all loads, though the relative gains are not as large as they were in Section 3.3.

3.5 Closeness to SRPT

It is interesting to investigate how far SIFT is from SRPT. To this end, we evaluate the overall average delay under SRPT using the formulas derived in [16] for the expected response time of a flow of size x. We also evaluate the short-flow, respectively long-flow, average delay under SRPT by averaging the expected response time of a flow of size x over the size distribution of short flows (Equation (4)), respectively long flows (Equation (3)).

Figure 6 plots the short-flow, long-flow and overall average delay under SIFT with FIFO queues, and SRPT, for the same parameters as before. The corresponding results for single-queue FIFO are also presented for comparison. SIFT is quite close to SRPT for the short flows and both are orders of magnitude better than FIFO. In terms of overall average delay, SIFT performs somewhere in between the two other schemes. For small ρ it is closer to SRPT, and for large ρ it is closer to FIFO. It is interesting that SRPT does a lot better than SIFT for long flows. This counter-intuitive result is actually expected for the following reason: under both schemes, long flows are only serviced after short flows, but among long flows, SIFT uses FIFO which yields a much larger average delay than SRPT. Figure 7 shows the corresponding plots for SIFT with PS queues, single-queue PS, and SRPT.

It is important to recall that the implementation complexity of SIFT is far less than that of SRPT. This comes at the cost of performance; one cannot reasonably expect the performance of SIFT to be comparable to that of SRPT for all flows.

[5] Notice that here we ignore the accumulation of low priority flows while the server works on high priority flows, hence the derived expression is approximate. For a way to take this into account, see Section 4.7 of [14].

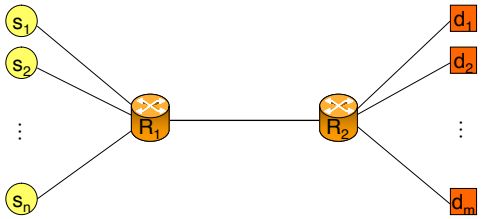

Figure 8: Network topology.

4. SIMULATIONS

This section evaluates the performance of SIFT in an Internet-like environment using ns-2 simulations [17]. Figure 8 shows the topology we use. There are n source nodes, m destination nodes, and two internal nodes, R_1 and R_2. The link capacity and the propagation delay between the source/destination nodes and internal nodes will vary from experiment to experiment. The capacity of the link between nodes R_1 and R_2 is 10Mbits/s and the propagation delay of this link is set to 0.1ms. SIFT is deployed on this link.

The two queues of SIFT and the strict priority mechanism between them is implemented using the CBQ functionality of ns-2. We also run simulations where the low priority queue is guaranteed a proportion of the link capacity. We call this scheme SIFT_{GBW}. Each of the two queues use either DropTail or RED. Their size is set to 100 packets. When SIFT is not used, the corresponding single queue has size equal to 200 packets. Throughout the experiments we use a sampling probability equal to 0.01.

The traffic is generated using the built-in sessions in ns-2. 300.000 web sessions are generated at random times at each experiment, each session consisting of a single object. This object is called a flow in the discussion below. All flows are transfered using TCP. Each flow consists of a Pareto-distributed number of packets with sizes varying between 1 and 10^8, and shape parameter equal to 1.1. (This is motivated by the well-known result that the size distribution of Internet flows is heavy-tailed, see, for example, [2, 21].) The packet size is set to 1000 bytes.

By varying the rate by which sources generate flows, we study the performance of SIFT at various levels of conges-

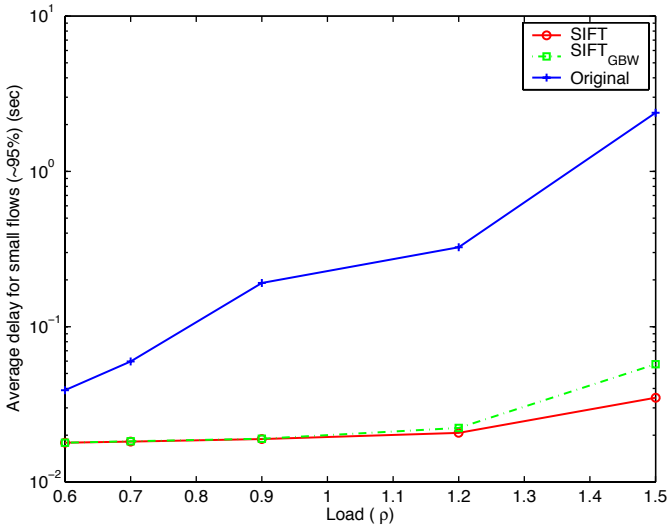

Figure 9: Average delay of short flows (95% of all flows) as a function of the load.

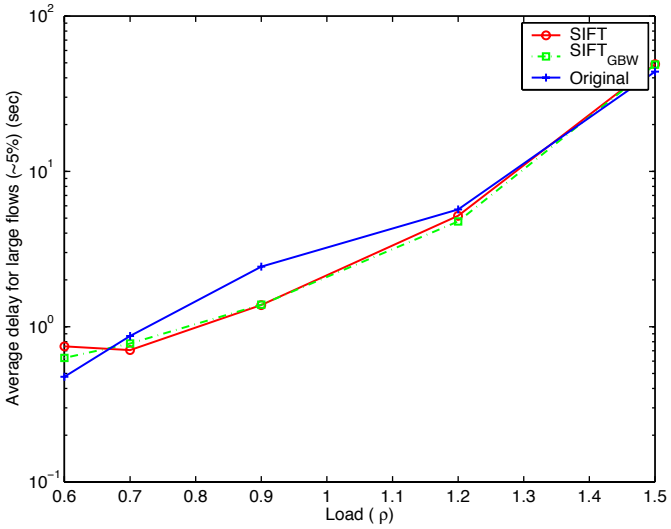

Figure 10: Average delay of large flows (5% of all flows) as a function of the load.

tion. In the rest of the section we characterize the congestion level of each experiment by the traffic intensity in the bottleneck link, defined as $\rho = \lambda E(S)/C$, where λ is the aggregate average arrival rate, $E(S)$ is the average flow size, and C is the link capacity. The percentage of drops in the experiments varies from 0 to 9% depending on the load.

4.1 Basic setup

We start with the simplest topology, where we have just one source destination pair ($n = m = 1$). The link capacities between the source/destination and the internal nodes are 100Mbits/s and the propagation delay of these links equals 1ms.[6]

We first present results when RED is used. In this experiment, we find that approximately 95% of all flows are "short", i.e., they are never sampled, and 5% of all flows are "long". (This fraction is, of course, a function of the sampling probability as well as the job size distribution; we have not attempted to optimize the sampling probability for the best average delays.) Figure 9 compares the average delay of the short flows with and without SIFT for $\rho = 0.6$, 0.7, 0.9, 1.2, and 1.5. It is evident from the plot that with SIFT, the short flows can have a delay close to two orders of magnitude less than without SIFT. Figure 10 shows that this gain does not significantly increase the delay of the rest of the flows: the average delay of long flows is at worst doubled under SIFT. Finally, Figure 11 compares the overall average delay of flows, and shows a 2x to 4x improvement, depending on the load.

Figure 12 shows the short-flow, long-flow, and overall average delays when the queues use DropTail instead of RED. The results show gains similar to the previous case.

Remark: It is worth mentioning that unequal partitioning of the total buffer space between the two queues might yield better performance than setting $B_h = B_l = B/2$. For example, because the high priority queue has strict priority, one expects its buffer space requirement to be relatively low, and there is no need to allocate $B/2$ buffer space to it.

Figure 13 plots the instantaneous queue sizes when DropTail is used and $\rho = 0.9$. It is evident that if the high priority queue had a size of 50 instead of 100, the results would have been identical. This is true not because the system is never congested; the original queue does get full at some point in time. For larger ρ, this is even more pronounced. For example, for $\rho = 1.2$, the original and the low priority queues are nearly always full, while the high priority queue has never more than 60 packets. Interestingly, this is a general trend in all experiments: the original 200-packet and the low priority 100-packet queue face similar levels of congestion, while the high priority 100-packet queue faces significantly lower congestion or no congestion at all. (Due to limitations of space we don't present more plots with queue dynamics.) Hence, it is plausible that in addition to significantly reducing average flow delays, SIFT also leads to sizable buffer savings at the packet level. Since the focus of this work is on reducing delays, we leave as future work to further investigate SIFT's potential for buffer savings.

4.2 Guaranteeing bandwidth for long flows

We also run simulations where a fraction of the total link bandwidth (10% in this particular experiment) is dedicated to the low priority queues containing the long flows. This ensures that long flows are not starved. Figures 9, 10 and 11 show that the performance of SIFT$_{GBW}$ is very similar to that of SIFT which uses strict priority. As expected, the average delays for long flows are smaller when bandwidth is guaranteed to the low priority queue, and the average delays for short flows are larger. It is evident that the difference between the delays is small enough to be negligible. So, at least on an average, we need not fear starvation for long flows with SIFT. Alternatively, if it is a concern, we could

[6] In order to better observe the effect of SIFT on queueing delay, we choose a small propagation delay here. This makes queueing delay the dominant component of total delay . Later in this section we also present results corresponding to larger propagation delays.

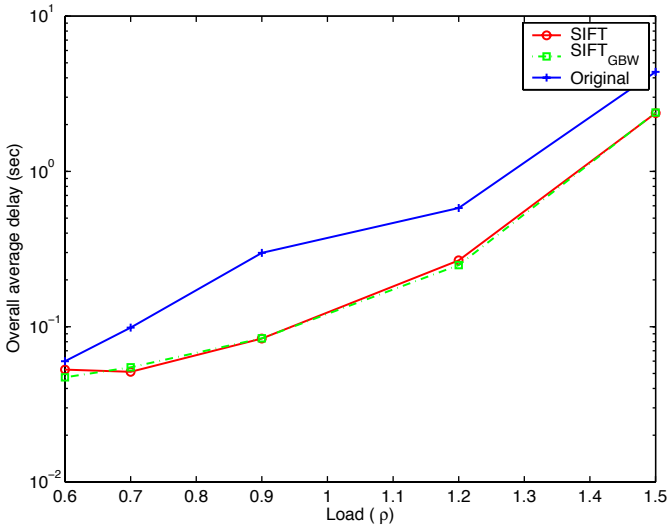

Figure 11: Overall average delay as a function of the load.

implement SIFT$_{GBW}$.

Note that under this scenario packet reordering is possible. This occurs when the last packet to join the high priority queue from a long flow, leaves the router after subsequent packets of the flow. However, this is very unlikely to occur since subsequent packets join the low priority queue, and this queue is not only larger on average, but also is serviced by a very small fraction of the total link capacity.

4.3 Multiple propagation delays

It is known [1] that when several large TCP flows share a link, bandwidth is shared between them in inverse proportion to their round trip times (RTTs). Given this, one worry might be that SIFT in conjunction with TCP would excessively penalize large flows with large RTTs, and drastically reduce their throughput. To investigate this, we run simulations with three source destination pairs ($n = m = 3$). For the first source-destination pair, we use a propagation delay of 0.1ms on the links from source/destination to the internal nodes; 1 ms on each link for the second, and 5 ms for the third pair. The speeds of these links are all 100Mbits/s.

It is clear from Figures 14, 15 and 16 that each of these sets of flows has lower average delays under SIFT. Of course, the values of these delays are higher for the flows with large RTTs than for the ones with small RTTs, but SIFT consistently provides gains for all three classes of flows.

4.4 Fast and slow flows

Suppose now that the following two kinds of flows share a link: *Turtles*, with low bit rates, and *Cheetahs*, with high bit rates. Note that it is possible to have a large turtle and a small cheetah; the former is a large-sized flow from which packets arrive at a small rate, while the latter is a small-sized high rate flow. One worry might be that in this situation SIFT would further slow down the already slow, large turtles, since the sampling mechanism here is packet based, and hence responds only to flow sizes without taking flow rates into account. A *temporal* (rather than a packet-based) sampling scheme, like the one suggested in [5], could be used if differentiation based on rate is also desirable. A combination of packet-based and temporal sampling can protect large turtles from being further slowed down. Another benefit of this setup is that it could identify small turtles whose rate is so slow that there is no delay gain from forwarding their packets in the high priority queue.

To study the effect of SIFT on average delays for turtles as well as cheetahs (in particular, to see how badly a packet-based sampling scheme would hurt large turtles), we use a simulation scenario with two source-destination pairs, both passing through the bottleneck link. One link, on which turtle flows arrive, has a speed of 5Mbits/s, and the other, on which cheetah flows arrive, runs at 495Mbits/s. The speed of the bottleneck link, as always, is 10Mbits/s. We ensure, by appropriately choosing the rate for the two sets of flows, that the turtles' link is never congested.

From Figure 17, it is clear that the average delays for turtles with SIFT is less than that without SIFT. That is, sharing a link with the cheetahs under SIFT does *not* increase the delays for the turtles, as feared earlier. Further, Figure 18 shows that the average delays for the cheetahs also reduces with SIFT, and the gain is greater for the cheetahs than for the turtles. In both cases, SIFT leads to lower average delays overall, and an order of magnitude improvement for most flows.

4.5 Many slow links

Finally, we consider the case where we have a large number of slow links feeding the bottleneck link. This is meant to capture the case, for example, when a large number of users connect to the Internet through slow modem links. We simulate this scenario using 10 source destination pairs, with each input and output link having a speed of 3Mbits/s. The bottleneck link has a speed of 10Mbits/s, as usual. In this case also, Figure 19 shows that using SIFT leads to significant improvements in performance for most flows, at the expense of only a small increase in delay for the large flows.

5. IMPLEMENTATION AND DEPLOYMENT

In this section we asess the implementation requirements of SIFT and comment on its deployability.

Compared to a traditional linecard, a SIFT-enabled linecard requires the following additions: (i) maintaining two queues instead of one in the physical buffer, (ii) implementing strict priority between the two queues, (iii) sampling packets at the time of their arrival, (iv) maintaining the flow id of the sampled packets, and (iv) evaluating whether a flow is sampled or not.

The first two requirements are clearly very simple. Maintaining two queues only requires a second linked list, and strict priority only requires to check if the head of the list corresponding to the high priority queue is null or not. Also, note that circular buffers may be used to make possible to "lend" unused buffer space to the other queue. The last requirement is also very easy to fulfill. Standard hashing schemes can be used to evaluate if a flow is sampled or not at a very low cost.

Sampling packets is also very inexpensive. A pseudo-random number generator can be used to decide whether to sample or not, or one may choose to "sample" every, say, 100^{th} packet. The later is a common practice to avoid random sampling; for example, it is used in Cisco's Netflow

monitoring tool [4].

Keeping the id of every sampled flow might be problematic if the number of sampled flows is too large. However, once a flow terminates, its flow id is removed. This keeps the number of active sampled flows small. In all our simulations this number was never more than a couple of thousand flows. More specifically, Table 1 reports the number of sampled flows (state size) for the simulation scenarios that were presented in Section 4 for various loads. The second column is the average state size over various loads (ρ varies from 0.6 to 1.5) and the third column is the maximum state size (achieved for $\rho = 1.5$ in all cases). Recall that in all the scenarios, the bottleneck link capacity equals 10Mbits/sec, and the sampling probability equals 0.01.

Experiment	Average state	Maximum state
Basic	797	1504
Various RTTs	812	1537
Fast and slow flows	755	1314
Many slow flows	806	1510

Table 1: State size (number of active sampled flows) in the various simulation scenarios.

A router that deploys SIFT should have a mechanism to identify which flow each packet belongs to. Such mechanisms are widely available today in commercial routers and many such routers operating at all but the fastest links have them activated. Notice that in accordance with usual practice [6, 8, 9], packets are said to belong to the same flow if they have the same source and destination IP address, and source and destination port number. A flow is "on" if its packets arrive more frequently than a timeout of some seconds. This timeout is usually set to something less than 60 seconds.

SIFT-enabled nodes should be placed before bottlenecks, for example, at access routers or wireless gateways. We expect one such node per path to be enough for reducing delays. More such nodes per path might penalize disproportionally long flows. Finally, a SIFT-enabled node cannot reside in places on the network where it is not possible to keep track of flows, for example, in front of very fast links at the core. Since these links are over provisioned and rarely congested, this is not a limitation.

6. CONCLUSIONS

Motivated by the heavy-tailed sizes of Internet flows, we propose SIFT, a scheme that behaves similarly to SRPT in the sense that it services shorter flows before longer flows. Using queueing theory, we show that the proposed scheme reduces average delay by orders of magnitude in comparison to FIFO or PS, and performs close to SRPT for a vast majority of flows.

The scheme is simple and easy to implement in practice. Extensive network simulations reveal that SIFT can reduce the average delay of short flows between one and two orders of magnitude, and the overall average delay between two and ten times, without increasing the delay of long flows by more than a factor of two or three.

Future work: We plan to perform simulations in more complex networks with multiple congested links, study the extend at which SIFT can lead to buffer savings, and investigate in more detail the implications of SIFT to congestion control.

7. REFERENCES

[1] E. Altman, C. Barakat, and K. Avrachenkov. A stochastic model of tcp/ip with stationary random losses. In *ACM SIGCOMM*, 2001.

[2] Sprint ATL. Sprint network traffic flow traces. `http://www.sprintlabs.com/Department/IP-Interworking/Monitor/`, accessed 2002.

[3] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: investigating unfairness. In *ACM SIGMETRICS/Performance*, pages 279–290, 2001.

[4] Cisco. NetFlow services and applications. White paper, 2000. `http://cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`, accessed January 2002.

[5] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, 2002.

[6] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Proceedings of the 4th Global Internet Symposium*, December 1999.

[7] A. Feldmann. Characteristics of TCP connection arrivals. In *Self-Similar Network Traffic and Performance Evaluation*, K.Park and W.Willinger, editors, Wiley, 2000.

[8] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Workshop on Passive and Active Measurements, PAM*, April 2001.

[9] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Packet-level traffic measurements from a tier-1 IP backbone. Technical Report TR01-ATL-110101, Sprint ATL Technical Report, November 2001.

[10] S. Ben Fredj, T. Bonalds, A. Prutiere, G. Gegnie, and J. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *Proceedings of SIGCOMM*, 2001.

[11] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal. Implementation of srpt scheduling in web servers. Technical Report CMU-CS-00-170, Carnegie Mellon Universit, 2000.

[12] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3), 1997.

[13] L. Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. Wiley, 1976.

[14] L. Kleinrock. *Queueing Systems. Volume II: Computer Applications*. John Wiley & Sons, 1976.

[15] W. Leland and T. Ott. Load-balancing heuristics and process behavior. In *Proceedings of the ACM SIGMETRICS Conference*, 1986.

[16] L.E.Schrage and L.W. Miller. The queue m/g/1 with the shortest processing remaining time discipline. *Operations Research*, 14:670–684, 1966.

[17] Network simulator. `http://www.isi.edu/nsnam/ns`, accessed June 2003.

[18] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack. Analysis of LAS scheduling for job size distributions with high variance. In *Proceedings of the ACM SIGMETRICS Conference*, 2003.

[19] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, May–June 1968.

[20] R. van den Heever. *Computer Time-sharing Priority Systems*. PhD thesis, University of California, Berkeley, 1969.

[21] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[22] R. W. Wolff. *Stochastic Networks and the Theory of Queues*. Prentice Hall, 1989.

[23] U. Madhow Z. Shao. Scheduling heavy-tailed traffic over the wireless internet. In *Proc. IEEE Vehicular Technology Conference*, September 2002.

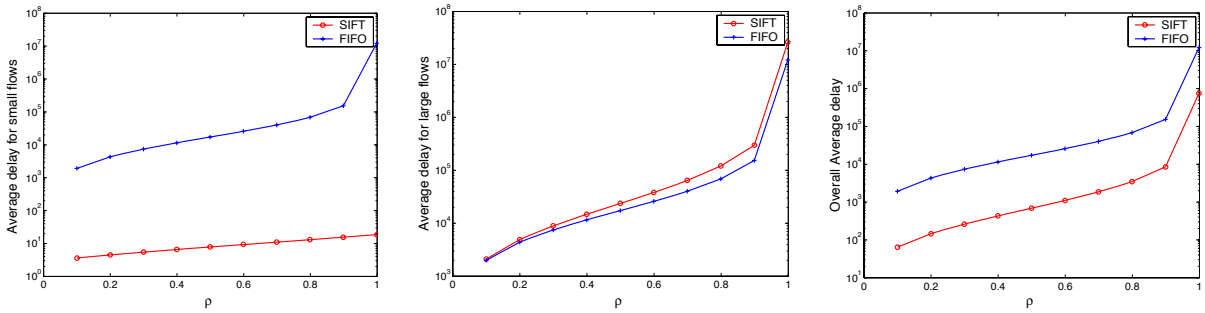

Figure 4: Short-flow, long-flow, and overall average delay of SIFT with FIFO queues, versus a single FIFO queue.

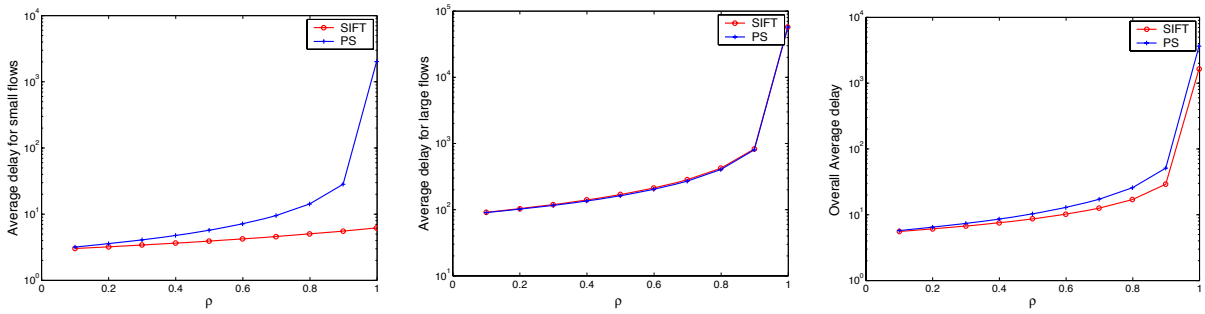

Figure 5: Short-flow, long-flow and overall average delay of SIFT with PS queues, versus a single PS queue.

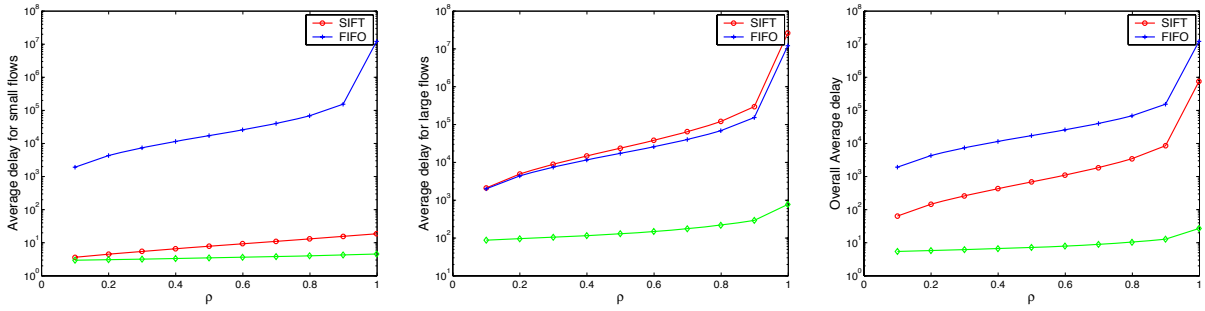

Figure 6: Short-flow, long-flow and overall average delay of SIFT with FIFO queues, single-queue FIFO, and SRPT.

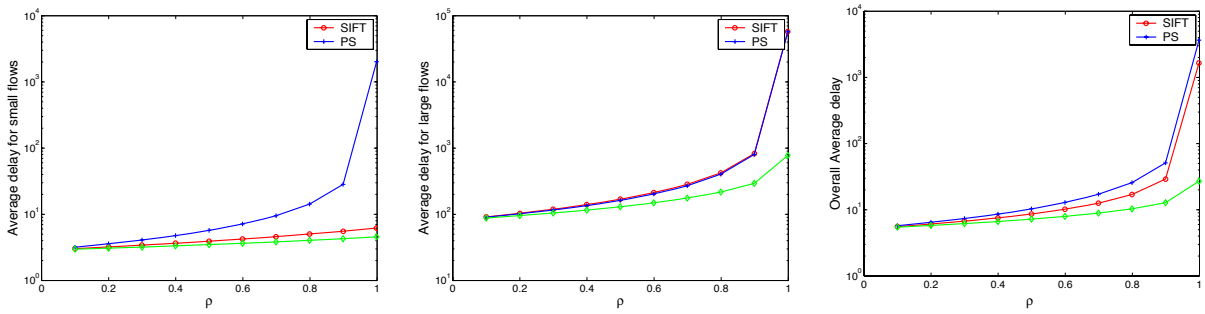

Figure 7: Short-flow, long-flow and overall average delay of SIFT with PS queues, single-queue PS, and SRPT.

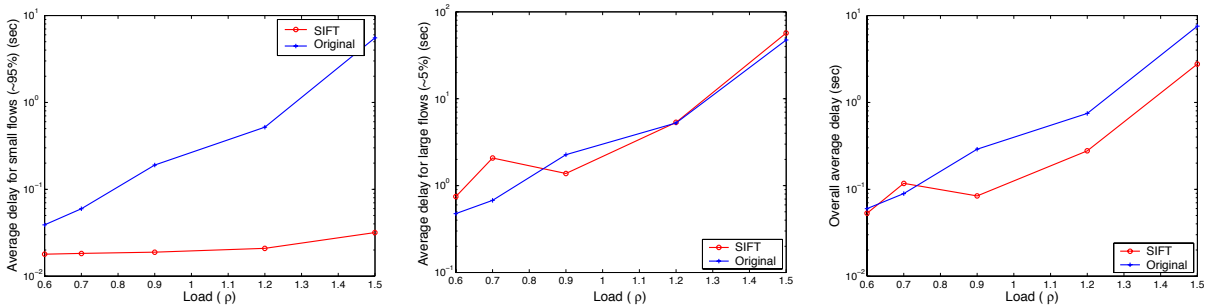

Figure 12: Average delays under SIFT for small, large and all flows with DropTail.

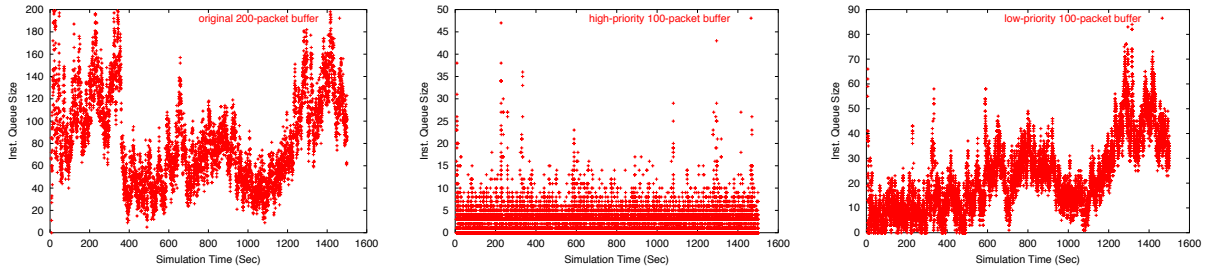

Figure 13: Instantaneous sizes of the original 200-packet buffer, the high-priority 100-packet buffer, and the low-priority 100-packet buffer.

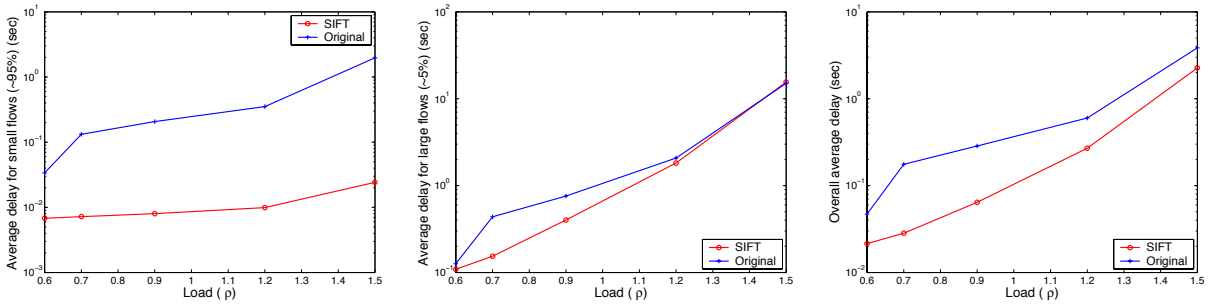

Figure 14: Average delays for small flows, large flows, and all flows for flows with an RTT of 0.6ms.

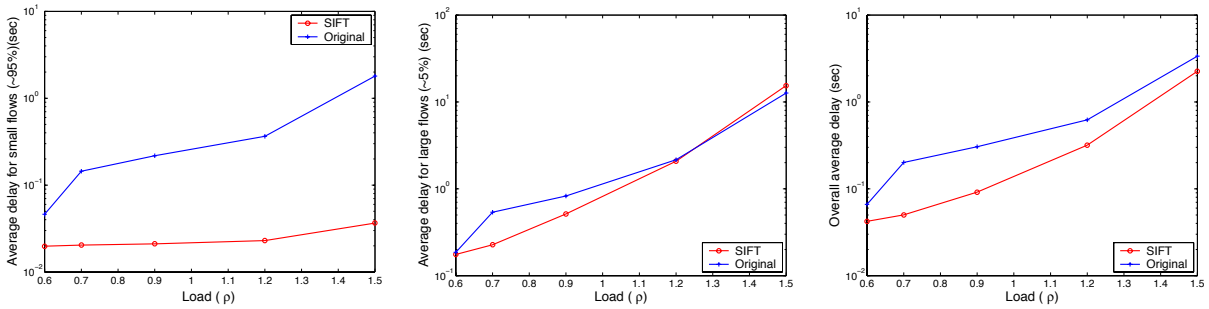

Figure 15: Average delays for small flows, large flows, and all flows with RTT = 4.2ms.

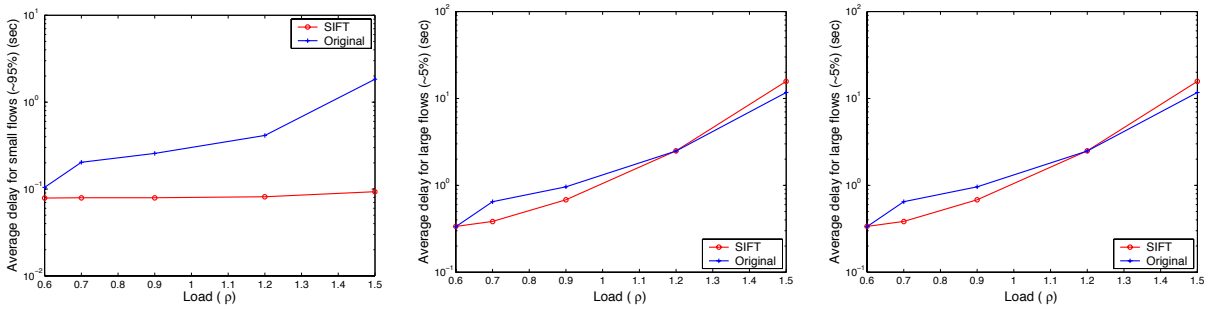

Figure 16: Average delays for small flows, large flows, and all flows with RTT = 20.2ms.

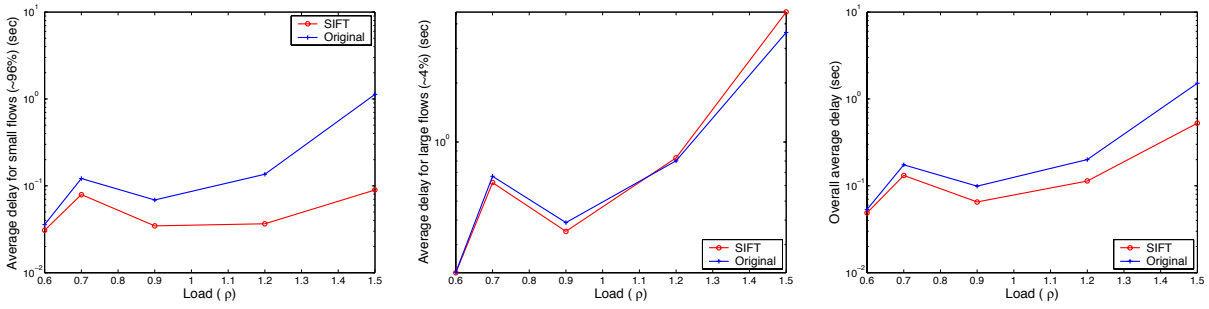

Figure 17: Average delays for small flows, large flows, and all flows for turtles (link speed 5Mbits/s).

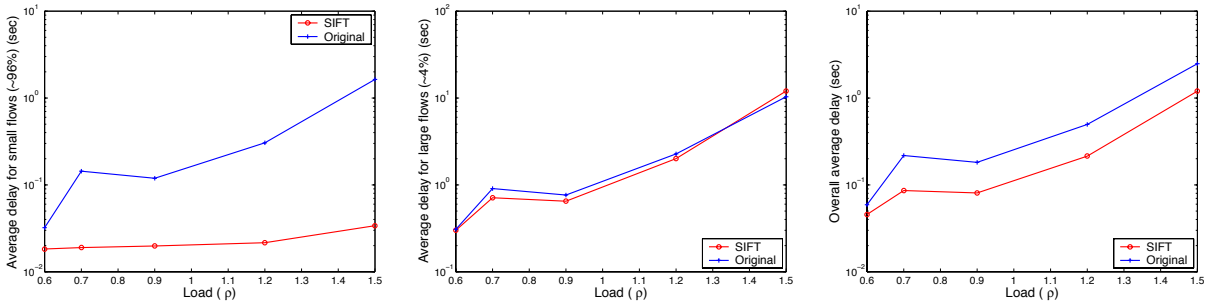

Figure 18: Average delays for small flows, large flows, and all flows for cheetahs (link speed 495 Mbits/s).

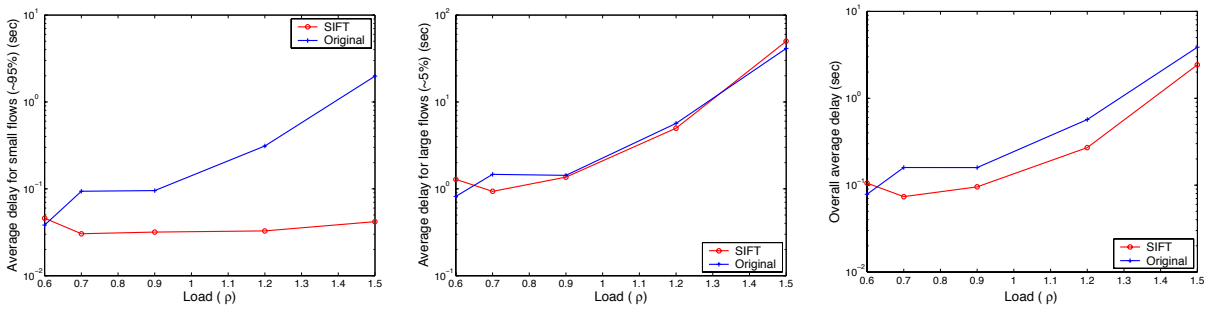

Figure 19: Average delays for small flows, large flows, and all flows (many slow links case).